UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Matheus Costa Monteiro

Análise comparativa de algoritmos de escalonamento de workflows científicos em processadores heterogêneos

Uberlândia, Brasil

2025

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Matheus Costa Monteiro

Análise comparativa de algoritmos de escalonamento de workflows científicos em processadores heterogêneos

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Sistemas de Informação.

Orientador: Paulo Henrique Ribeiro Gabriel

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Sistemas de Informação

Uberlândia, Brasil

2025

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
Faculdade de Computação
Av. João Naves de Ávila, nº 2121, Bloco 1A - Bairro Santa Mônica, Uberlândia-MG, CEP 38400-902
Telefone: (34) 3239-4144 - http://www.portal.facom.ufu.br/ facom@ufu.br

ATA DE DEFESA - GRADUAÇÃO

Curso de Graduação em:	Bacharelado em Sistemas de Informação				
Defesa de:	FACOM31802 - Trabalho de Conclusão de Curso 2				
Data:	23/09/2025	Hora de início:	10:30	Hora de encerramento:	10:45
Matrícula do Discente:	12111BSI281				
Nome do Discente:	Matheus Costa Monteiro				
Título do Trabalho:	Análise comparativa de algoritmos de escalonamento de workflows científicos em processadores heterogêneos				
A carga horária curricular foi cumprida integralmente?	(X) Sim ( ) Não				

Reuniu-se remotamente por meio da plataforma Microsoft Teams, da Universidade Federal de Uberlândia, a Banca Examinadora, designada pelo Colegiado do Curso de Graduação em Sistemas de Informação, assim composta: Professores: Dr. Rafael Dias Araújo - FACOM/UFU; Dr. Rodrigo Sanches Miani - FACOM/UFU; e Dr. Paulo Henrique Ribeiro Gabriel - FACOM/UFU, orientador do candidato.

Iniciando os trabalhos, o presidente da mesa, Dr. Paulo Henrique Ribeiro Gabriel, apresentou a Comissão Examinadora e o candidato, agradeceu a presença do público, e concedeu ao discente a palavra, para a exposição do seu trabalho. A duração da apresentação do discente e o tempo de arguição e resposta foram conforme as normas do curso.

A seguir o senhor presidente concedeu a palavra, pela ordem sucessivamente, aos examinadores, que passaram a arguir o candidato. Ultimada a arguição, que se desenvolveu dentro dos termos regimentais, a Banca, em sessão secreta, atribuiu o resultado final, considerando o candidato:

(X) Aprovado Nota [100] (Somente números inteiros)

Nada mais havendo a tratar foram encerrados os trabalhos. Foi lavrada a presente ata que após lida e achada conforme foi assinada pela Banca Examinadora.

Documento assinado eletronicamente por **Paulo Henrique Ribeiro Gabriel**, **Professor(a) do Magistério Superior**, em 23/09/2025, às 10:47, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Rodrigo Sanches Miani**, **Professor(a) do Magistério Superior**, em 23/09/2025, às 10:48, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Rafael Dias Araújo**, **Professor(a) do Magistério Superior**, em 23/09/2025, às 11:17, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site https://www.sei.ufu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **6701664** e o código CRC **5E063587**.

Agradecimentos

Agradeço, primeiramente, a Deus, que me concedeu saúde, inteligência e perseverança ao longo de toda a minha jornada na graduação. À Nossa Senhora da Abadia, por guiar meus passos e abençoar meus estudos e projetos.

Ao meu pai, **Alessandro Monteiro de Araújo**, que foi e sempre será minha inspiração máxima na área de computação e em toda minha vida, pelo apoio incondicional para meu crescimento acadêmico e pessoal. À minha mãe, **Euzania Luzia da Costa Araújo**, por todo amor e por sempre acreditar nas minhas ideias e projetos.

À minha querida **Amanda**, por confiar, incentivar e acreditar em mim em todos os momentos. Aos meus caros amigos e família, que me acompanharam nesta jornada e estiveram presentes nos desafios e nas conquistas.

Por fim, agradeço ao meu orientador, **Prof.Dr. Paulo Henrique Ribeiro Gabriel**, por todo o conhecimento compartilhado e pela paciência e dedicação durante esta caminhada, e a todos os professores e mentores que contribuíram para minha formação.

Resumo

Em computação de alto desempenho (HPC), algoritmos de escalonamento são decisivos para transformar hardware em desempenho, custo e eficiência energética — aspecto crítico em cargas modernas que incluem o treinamento de modelos de **aprendizado de máquina**, cada vez mais intensivas em recursos. Diferentemente de ambientes *homogêneos* (nós idênticos), *clusters heterogêneos* de CPUs/GPUs exibem variação de capacidade de processamento e de comunicação; assim, a política de mapeamento e ordenação de tarefas torna-se central para minimizar filas, latências e ociosidade, reduzindo o tempo de execução, o consumo de energia e, consequentemente, a pegada de CO_2.

Neste trabalho foi comparado seis heurísticas clássicas de *list scheduling* — **DLS**, **HEFT**, **HEFT-LA**, **PEFT**, **IHEFT** e **IPEFT** — em um *pipeline* reprodutível *simulado*, padronizando a interface de E/S, o gerador de DAGs e o cálculo de métricas para isolar o efeito da lógica de decisão. Avaliamos instâncias de grande porte (256–4096 tarefas) derivadas de *workflows* científicos, em conjuntos de 8, 16 e 32 máquinas, medindo *makespan*, balanceamento de carga, *waiting time* e energia.

Os resultados mostram que heurísticas com *look-ahead* (**PEFT**, **IPEFT**, **HEFT-LA**) reduzem consistentemente *makespan* e *waiting time* em cenários com comunicação dominante; **PEFT** e **IPEFT** preservam a complexidade $O(V^2P)$, ao passo que o **HEFT-LA** apresenta complexidade assintótica superior. Sob baixa comunicação, **HEFT** permanece competitivo. **IHEFT** e **IPEFT** mitigam lacunas de escalonamento e elevam a utilização. Observamos um compromisso energia–desempenho: menor ociosidade tende a reduzir a energia total, mesmo com leve aumento da potência média. Além disso, o **DLS** apresenta ganhos de velocidade com o aumento de máquinas ao capturar paralelismo de forma mais *compute-bound*, enquanto heurísticas com predição podem tornar-se mais sensíveis às barreiras de comunicação. Como artefatos, disponibilizamos implementações *paper-faithful*, um gerador de casos com controle de semente e um *pipeline* para comparações futuras em ambientes heterogêneos.

Palavras-chave: Computação de Alto Desempenho; *Workflows* científicos; Escalonamento de tarefas; DAG; Modelos de ML.

Abstract

In high-performance computing (HPC), scheduling algorithms are decisive to translate hardware into performance, cost efficiency, and energy efficiency—an especially critical concern for modern workloads such as **machine learning** training. Unlike *homogeneous* settings (identical nodes), heterogeneous CPU/GPU clusters vary in compute and communication capabilities; thus, task mapping and ordering policies become central to minimizing queues, latency, and idleness, thereby reducing runtime, energy, and the CO_2 footprint.

This work compares six classic *list scheduling* heuristics—**DLS**, **HEFT**, **HEFT-LA**, **PEFT**, **IHEFT**, and **IPEFT**—within a fully reproducible *simulated* pipeline that standardizes I/O interfaces, DAG generation, and metric computation to isolate decision-logic effects. We evaluate large workflow-derived instances (256–4096 tasks) on sets of 8, 16, and 32 machines, measuring *makespan*, load balance, *waiting time*, and energy.

Results indicate that look-ahead heuristics (**PEFT**, **IPEFT**, **HEFT-LA**) consistently reduce makespan and waiting time in communication-dominated scenarios; **PEFT** and **IPEFT** preserve $O(V^2P)$ time complexity, whereas **HEFT-LA** incurs higher asymptotic cost. Under low communication, **HEFT** remains competitive. **IHEFT** and **IPEFT** mitigate scheduling gaps and raise utilization. We observe an energy–performance trade-off: curbing idleness often lowers total energy even with a slight rise in average power. Additionally, **DLS** achieves speedups as machine count grows by exploiting compute-bound parallelism, while predictive heuristics become more sensitive to communication bottlenecks. We release paper-faithful implementations, a seeded case generator, and a comparison pipeline to support future studies in heterogeneous environments.

Keywords: High-Performance Computing; Scientific workflows; Task scheduling; DAG; Machine Learning models.

Lista de ilustrações

Figura 1 – Classificação de algoritmos *estáticos* de escalonamento: (i) *Heuristic-based* (subdivididos em *List Scheduling*, *Clustering* e *Task Duplication*) e (ii) *Guided Random Search-based* (e.g., AG, *simulated annealing*, busca local). 19
Figura 2 – Exemplo de modelagem de aplicação como um DAG de tarefas (arestas indicam precedências e fluxo de dados). 27
Figura 3 – DAG ilustrativo do cenário 1 (N=256). 54
Figura 4 – Métricas (MS, $\bar{U}$, $\overline{\mathrm{WT}}$, E_{total}) – cenário 1, P=8. 55
Figura 5 – Gantt – DLS, cenário 1, P=8. 55
Figura 6 – Gantt – HEFT, cenário 1, P=8. 56
Figura 7 – Gantt – HEFT-LA, cenário 1, P=8. 56
Figura 8 – Gantt – PEFT, cenário 1, P=8. 56
Figura 9 – Gantt – IHEFT, cenário 1, P=8. 56
Figura 10 – Gantt – IPEFT, cenário 1, P=8. 57
Figura 11 – Métricas (MS, $\bar{U}$, $\overline{\mathrm{WT}}$, E_{total}) – cenário 1, P=16. 58
Figura 12 – Gantt – DLS, cenário 1, P=16. 59
Figura 13 – Gantt – HEFT, cenário 1, P=16. 59
Figura 14 – Gantt – HEFT-LA, cenário 1, P=16. 60
Figura 15 – Gantt – PEFT, cenário 1, P=16. 60
Figura 16 – Gantt – IHEFT, cenário 1, P=16. 61
Figura 17 – Gantt – IPEFT, cenário 1, P=16. 61
Figura 18 – Métricas (MS, $\bar{U}$, $\overline{\mathrm{WT}}$, E_{total}) – cenário 1, P=32. 62
Figura 19 – Gantt – DLS, cenário 1, P=32. 63
Figura 20 – Gantt – HEFT, cenário 1, P=32. 64
Figura 21 – Gantt – HEFT-LA, cenário 1, P=32. 65
Figura 22 – Gantt – PEFT, cenário 1, P=32. 66
Figura 23 – Gantt – IHEFT, cenário 1, P=32. 67
Figura 24 – Gantt – IPEFT, cenário 1, P=32. 68
Figura 25 – Síntese visual do cenário 1: evolução das métricas com $P \in \{8, 16, 32\}$. 69
Figura 26 – DAG ilustrativo do cenário 2 (N=512). 69
Figura 27 – Métricas (MS, $\bar{U}$, $\overline{\mathrm{WT}}$, E_{total}) – cenário 2, P=8. 70
Figura 28 – Gantt – DLS, cenário 2, P=8. 70
Figura 29 – Gantt – HEFT, cenário 2, P=8. 71
Figura 30 – Gantt – HEFT-LA, cenário 2, P=8. 71
Figura 31 – Gantt – PEFT, cenário 2, P=8. 71
Figura 32 – Gantt – IHEFT, cenário 2, P=8. 71

Figura 33 – Gantt – IPEFT, cenário 2, P=8. 72
Figura 34 – Métricas (MS, $\bar{U}$, $\overline{\mathrm{WT}}$, E_{total}) – cenário 2, P=16. 73
Figura 35 – Gantt – DLS, cenário 2, P=16. 74
Figura 36 – Gantt – HEFT, cenário 2, P=16. 74
Figura 37 – Gantt – HEFT-LA, cenário 2, P=16. 75
Figura 38 – Gantt – PEFT, cenário 2, P=16. 75
Figura 39 – Gantt – IHEFT, cenário 2, P=16. 76
Figura 40 – Gantt – IPEFT, cenário 2, P=16. 76
Figura 41 – Métricas (MS, $\bar{U}$, $\overline{\mathrm{WT}}$, E_{total}) – cenário 2, P=32. 77
Figura 42 – Gantt – DLS, cenário 2, P=32. 78
Figura 43 – Gantt – HEFT, cenário 2, P=32. 79
Figura 44 – Gantt – HEFT-LA, cenário 2, P=32. 80
Figura 45 – Gantt – PEFT, cenário 2, P=32. 81
Figura 46 – Gantt – IHEFT, cenário 2, P=32. 82
Figura 47 – Gantt – IPEFT, cenário 2, P=32. 83
Figura 48 – Síntese visual do cenário 2: evolução das métricas com $P \in \{8, 16, 32\}$. 84
Figura 49 – DAG ilustrativo do cenário 3 (N=1024). 84
Figura 50 – Métricas (MS, $\bar{U}$, $\overline{\mathrm{WT}}$, E_{total}) — cenário 3, P=8. 85
Figura 51 – Gantt — DLS, cenário 3, P=8. 85
Figura 52 – Gantt — HEFT, cenário 3, P=8. 86
Figura 53 – Gantt — HEFT-LA, cenário 3, P=8. 86
Figura 54 – Gantt — PEFT, cenário 3, P=8. 86
Figura 55 – Gantt — IHEFT, cenário 3, P=8. 86
Figura 56 – Gantt — IPEFT, cenário 3, P=8. 87
Figura 57 – Métricas (MS, $\bar{U}$, $\overline{\mathrm{WT}}$, E_{total}) — cenário 3, P=16. 87
Figura 58 – Gantt – DLS, cenário 3, P=16. 88
Figura 59 – Gantt – HEFT, cenário 3, P=16. 88
Figura 60 – Gantt – HEFT-LA, cenário 3, P=16. 89
Figura 61 – Gantt – PEFT, cenário 3, P=16. 89
Figura 62 – Gantt – IHEFT, cenário 3, P=16. 90
Figura 63 – Gantt – IPEFT, cenário 3, P=16. 90
Figura 64 – Métricas (MS, $\bar{U}$, $\overline{\mathrm{WT}}$, E_{total}) – cenário 3, P=32. 91
Figura 65 – Gantt – DLS, cenário 3, P=32. 92
Figura 66 – Gantt – HEFT, cenário 3, P=32. 93
Figura 67 – Gantt – HEFT-LA, cenário 3, P=32. 94
Figura 68 – Gantt – PEFT, cenário 3, P=32. 95
Figura 69 – Gantt – IHEFT, cenário 3, P=32. 96
Figura 70 – Gantt – IPEFT, cenário 3, P=32. 97
Figura 71 – Síntese visual do cenário 3: evolução das métricas com $P \in \{8, 16, 32\}$. 98

Figura 72 – DAG ilustrativo do cenário 4 (N=2048). 98
Figura 73 – Métricas (MS, $\bar{U}$, $\overline{\mathrm{WT}}$, E_{total}) – cenário 4, P=8. 99
Figura 74 – Gantt – DLS, cenário 4, P=8. 99
Figura 75 – Gantt – HEFT, cenário 4, P=8. 100
Figura 76 – Gantt – HEFT-LA, cenário 4, P=8. 100
Figura 77 – Gantt – PEFT, cenário 4, P=8. 100
Figura 78 – Gantt – IHEFT, cenário 4, P=8. 100
Figura 79 – Gantt – IPEFT, cenário 4, P=8. 101
Figura 80 – Métricas (MS, $\bar{U}$, $\overline{\mathrm{WT}}$, E_{total}) — cenário 4, P=16. 101
Figura 81 – Gantt — DLS, cenário 4, P=16. 102
Figura 82 – Gantt — HEFT, cenário 4, P=16. 102
Figura 83 – Gantt — HEFT-LA, cenário 4, P=16. 103
Figura 84 – Gantt — PEFT, cenário 4, P=16. 103
Figura 85 – Gantt — IHEFT, cenário 4, P=16. 104
Figura 86 – Gantt — IPEFT, cenário 4, P=16. 104
Figura 87 – Métricas (MS, $\bar{U}$, $\overline{\mathrm{WT}}$, E_{total}) – cenário 4, P=32. 105
Figura 88 – Gantt – DLS, cenário 4, P=32. 106
Figura 89 – Gantt – HEFT, cenário 4, P=32. 107
Figura 90 – Gantt – HEFT-LA, cenário 4, P=32. 108
Figura 91 – Gantt – PEFT, cenário 4, P=32. 109
Figura 92 – Gantt – IHEFT, cenário 4, P=32. 110
Figura 93 – Gantt – IPEFT, cenário 4, P=32. 111
Figura 94 – Síntese visual do cenário 4: evolução das métricas com $P \in \{8, 16, 32\}$. 112
Figura 95 – DAG ilustrativo do cenário 5 (N=4096). 112
Figura 96 – Métricas (MS, $\bar{U}$, $\overline{\mathrm{WT}}$, E_{total}) – cenário 5, P=8. 113
Figura 97 – Gantt – DLS, cenário 5, P=8. 113
Figura 98 – Gantt – HEFT, cenário 5, P=8. 114
Figura 99 – Gantt – HEFT-LA, cenário 5, P=8. 114
Figura 100 – Gantt – PEFT, cenário 5, P=8. 114
Figura 101 – Gantt – IHEFT, cenário 5, P=8. 114
Figura 102 – Gantt – IPEFT, cenário 5, P=8. 115
Figura 103 – Métricas (MS, $\bar{U}$, $\overline{\mathrm{WT}}$, E_{total}) – cenário 5, P=16. 115
Figura 104 – Gantt – DLS, cenário 5, P=16. 116
Figura 105 – Gantt – HEFT, cenário 5, P=16. 116
Figura 106 – Gantt – HEFT-LA, cenário 5, P=16. 117
Figura 107 – Gantt – PEFT, cenário 5, P=16. 117
Figura 108 – Gantt – IHEFT, cenário 5, P=16. 118
Figura 109 – Gantt – IPEFT, cenário 5, P=16. 118
Figura 110 – Métricas (MS, $\bar{U}$, $\overline{\mathrm{WT}}$, E_{total}) – cenário 5, P=32. 119

Figura 111 – Gantt – DLS, cenário 5, P=32. 120
Figura 112 – Gantt – HEFT, cenário 5, P=32. 121
Figura 113 – Gantt – HEFT-LA, cenário 5, P=32. 122
Figura 114 – Gantt – PEFT, cenário 5, P=32. 123
Figura 115 – Gantt – IHEFT, cenário 5, P=32. 124
Figura 116 – Gantt – IPEFT, cenário 5, P=32. 125
Figura 117 – Síntese visual do cenário 5: evolução das métricas com $P \in \{8, 16, 32\}$. 126

Lista de tabelas

Tabela 1 – Parâmetros padrão das instâncias PEFT-like. 50
Tabela 2 – Cenários de experimentos 53

Lista de abreviaturas e siglas

CCR	Compute and Communication Ratio
CNP	Critical Node Parent
CNCT	Critical-Node-aware Cost Table
CPU	Central Processing Unit
DAG	Directed Acyclic Graph
DLS	Dynamic Level Scheduling
DP	Data Parallelism
EFT	Earliest Finish Time
EST	Earliest Start Time
GPU	Graphics Processing Unit
HEFT	Heterogeneous Earliest Finish Time
HEFT-LA	HEFT with Lookahead
IHEFT	Improved HEFT
IPEFT	Improved PEFT
LLM	Large Language Model
NVLink	NVIDIA High-Speed GPU Interconnect
OCT	Optimistic Cost Table
PCIe	Peripheral Component Interconnect Express
PCT	Pessimistic Cost Table
PEFT	Predict Earliest Finish Time
PP	Pipeline Parallelism
SLR	Schedule Length Ratio

Sumário

1 INTRODUÇÃO . . . 17
1.1 Contextualização do problema de escalonamento em sistemas heterogêneos . . . 18
1.1.1 Modelagem do problema . . . 20
1.2 Motivação deste trabalho. . . . 21
1.2.1 Metodologia . . . 21
1.3 Objetivo e contribuições . . . 22
1.4 Implementação e Ambiente Experimental . . . 23
1.4.1 Implementação . . . 23
1.4.1.0.1 Detalhes complementares . . . 24
1.4.2 Modelo de Entrada e Métricas . . . 24
1.4.3 Ambiente de Execução e Hardware . . . 24
1.4.4 Boas Práticas de Performance . . . 25

2 FUNDAMENTAÇÃO TEÓRICA . . . 26
2.1 Modelo de Tarefas (DAG) . . . 26
2.1.1 Atomicidade e não preempção . . . 26
2.2 Ambiente Heterogêneo e Custos . . . 27
2.3 Cronograma e Tempos (EST e EFT) . . . 27
2.4 Parâmetros de Entrada . . . 28
2.5 Métricas de Avaliação . . . 28
2.5.1 Makespan . . . 28
2.5.2 Balanceamento de carga (*Load Balancing*) . . . 28
2.5.3 Tempo médio de espera (*Waiting Time*) . . . 29
2.5.4 Custo de energia (pós-execução) . . . 29
2.6 Revisão Bibliográfica . . . 29
2.7 Trabalhos relacionados . . . 31
2.8 Considerações Finais . . . 31

3 DESENVOLVIMENTO . . . 32
3.1 Algoritmos Selecionados . . . 32
3.1.1 Dynamic Level Scheduling (DLS) . . . 33
3.1.2 Heterogeneous Earliest Finish Time (HEFT) . . . 35
3.1.3 Lookahead Scheduling (*HEFT-LA*) . . . 38
3.1.4 Predict Earliest Finish Time (PEFT) . . . 40
3.1.5 Improved HEFT (IHEFT) . . . 43

3.1.6 Improved PEFT (IPEFT) . . . 45
3.2 Geração Sintética de Grafos de Tarefas . . . 47
3.2.1 Forma do DAG . . . 47
3.2.2 Pesos de comunicação (arestas) . . . 48
3.2.3 Tempos de execução (heterogeneidade com afinidade) . . . 48
3.2.4 Potência/Energia (métrica derivada, não-decisória) . . . 48
3.2.5 Matriz de largura de banda (recursos) . . . 49
3.2.6 Parâmetros padrão . . . 50
3.2.7 Pseudocódigo (alto nível) . . . 50
3.2.8 Reprodutibilidade e sanidade . . . 50
3.2.9 Ajuste opcional de CCR . . . 51
3.2.10 Integração com os algoritmos . . . 51
3.2.10.0.1 Disponibilização e licença (MIT). . . . 51
3.2.10.0.2 Observação. . . . 51
3.2.10.1 Síntese . . . 51
3.3 Herança nas tabelas de custos . . . 52

4 EXPERIMENTOS . . . 53
4.1 Resultados por cenário . . . 53
4.2 Cenário 1: $N = 256$ tarefas . . . 54
4.2.1 Configuração com $P = 8$ . . . 54
4.2.1.1 Análise P=8 . . . 54
4.2.1.2 Síntese P=8 . . . 57
4.2.2 Configuração com $P = 16$ . . . 59
4.2.2.1 Análise P=16 . . . 59
4.2.2.2 Síntese P=16 . . . 61
4.2.3 Configuração com $P = 32$ . . . 62
4.2.3.1 Análise P=32 . . . 62
4.2.3.2 Síntese P=32 . . . 64
4.2.4 Síntese do cenário 1 (N=256) . . . 65
4.3 Cenário 2: $N = 512$ tarefas . . . 66
4.3.1 Configuração com $P = 8$ . . . 67
4.3.1.1 Análise P=8 . . . 67
4.3.1.2 Síntese P=8 . . . 68
4.3.2 Configuração com $P = 16$ . . . 72
4.3.2.1 Análise P=16 . . . 72
4.3.2.2 Síntese P=16 . . . 72
4.3.3 Configuração com $P = 32$ . . . 74
4.3.3.1 Análise P=32 . . . 74
4.3.3.2 Síntese P=32 . . . 75

4.3.4 Síntese do cenário 2 (N=512) . . . 76
4.4 Cenário 3: $N = 1024$ tarefas . . . 80
4.4.1 Configuração com $P = 8$ . . . 80
4.4.1.1 Análise P=8 . . . 80
4.4.1.2 Síntese P=8 . . . 82
4.4.2 Configuração com $P = 16$ . . . 83
4.4.2.1 Análise P=16 . . . 83
4.4.2.2 Síntese P=16 . . . 89
4.4.3 Configuração com $P = 32$ . . . 90
4.4.3.1 Análise P=32 . . . 90
4.4.3.2 Síntese P=32 . . . 93
4.4.4 Síntese do cenário 3 (N=1024) . . . 93
4.5 Cenário 4: $N = 2048$ tarefas . . . 95
4.5.0.1 Configuração com $P = 8$ . . . 95
4.5.0.2 Análise P=8 . . . 95
4.5.0.3 Configuração com $P = 16$ . . . 96
4.5.0.4 Análise P=16 . . . 96
4.5.0.5 Configuração com $P = 32$ . . . 97
4.5.0.6 Análise P=32 . . . 97
4.5.1 Síntese do cenário 4 . . . 104
4.5.1.0.1 Principais achados. . . . 104
4.5.1.0.2 Leituras cruzadas (sem comparar valores absolutos). . . . 107
4.6 Cenário 5: $N = 4096$ tarefas . . . 107
4.6.1 Configuração com $P = 8$ . . . 108
4.6.1.1 Análise P=8 . . . 108
4.6.2 Configuração com $P = 16$ . . . 109
4.6.2.1 Análise P=16 . . . 109
4.6.3 Configuração com $P = 32$ . . . 110
4.6.3.1 Análise P=32 . . . 110
4.6.3.2 Síntese P=32 . . . 111
4.6.4 Síntese do cenário 5 (N=4096) . . . 111

5 CONCLUSÃO . . . 127

REFERÊNCIAS . . . 130

APÊNDICES 133

APÊNDICE A – REPRODUTIBILIDADE DO PROJETO E BENCHMARK UNIFICADO 134

APÊNDICE B – TRABALHOS CORRELATOS NÃO INCLUÍDOS NA AVALIAÇÃO 136

B.1 CEFT – Constrained Earliest Finish Time (2012) 136

B.2 PATS – Predict and Arrange Task Scheduling (2016) 136

B.3 DVR-HEFT e CCATSF – variantes dinâmicas (2019) 136

B.4 HexiScale – treinamento de LLMs em GPUs heterogêneas (2025) . 137

B.4.0.0.1 Observação metodológica. 137

B.4.0.0.2 Roteiro para trabalhos futuros. 137

ANEXOS 138

ANEXO A – MANIFESTO DA RELEASE E LICENÇAS 139

A.1 Licenciamento do repositório 139

1 Introdução

Na Computação Científica, a modelagem e a execução de experimentos complexos são frequentemente organizadas por meio de *workflows*, que estruturam conjuntos de tarefas interdependentes em etapas sucessivas de processamento, comunicação e análise (BRAGHETTO; CORDEIRO, 2014). Essa abordagem, além de favorecer a reprodutibilidade e a automação, explicita a relevância de estratégias eficazes de alocação de recursos, uma vez que diferentes etapas de um *workflow* podem demandar capacidades computacionais heterogêneas. No contexto do treinamento de Modelos de Linguagem Grandes (LLMs), os *workflows* científicos assumem papel central, pois evidenciam o desafio de distribuir de maneira eficiente as tarefas entre clusters compostos por GPUs de diferentes gerações, memórias e interconexões, de modo a otimizar métricas como *makespan*, custo de execução e consumo energético.

Nesse contexto, o crescimento acelerado das aplicações de Inteligência Artificial (IA), em particular dos Modelos de Linguagem Grandes (*Large Language Models*, LLMs), como GPTs e LLaMAs, tem impulsionado a demanda por capacidade computacional. Esses modelos, devido à sua complexidade e ao volume de dados, demandam sistemas computacionais com múltiplos processadores gráficos (GPUs) e dispositivos de diferentes gerações, memórias e interconexões. Nesses cenários, *como* distribuir tarefas eficientemente entre recursos heterogêneos torna-se decisivo para reduzir *makespan*, custo e consumo energético.

Trabalhos recentes sugerem que a heterogeneidade pode ser explorada a favor do desempenho no treinamento de LLMs: arquiteturas que dividem assimetricamente carga e comunicação entre GPU ou /CPUs com diferentes potências e bandas de interconexão têm alcançado utilização de cluster comparável a sistemas homogêneos de alto desempenho (YAN et al., 2024). Em ambientes acadêmicos e industriais, é comum a agregação oportunista de laboratórios ou servidores com placas variadas, compondo as chamadas "fazendas" de GPUs mistas para treino e *fine-tuning*. Esses casos reforçam a necessidade de algoritmos de escalonamento sensíveis à heterogeneidade.

Recentemente, a literatura sobre treinamento distribuído de LLMs tem explorado explicitamente a execução sobre sistemas computacionais heterogêneos, combinando GPUs de diferentes gerações e capacidades (bem como CPUs de apoio) para aumentar a eficiência e a disponibilidade de recursos. Um exemplo representativo é o *HexiScale*, que propõe estratégias para acomodar o treinamento de modelos de linguagem em ambientes heterogêneos, mitigando desbalanceamentos e gargalos decorrentes da diversidade de hardware (YAN et al., 2025). Em linha semelhante, abordagens como o *FlashFlex*

investigam mecanismos de adaptação do treinamento a variações de desempenho entre dispositivos (YAN et al., 2024). Esse contexto é particularmente frequente na academia, onde laboratórios e centros de pesquisa reúnem GPUs e CPUs mais novas e mais antigas em um único *cluster* computacional, resultando naturalmente em ambientes heterogêneos; tal configuração é amplamente assumida e modelada nos estudos clássicos de escalonamento (ARABNEJAD; BARBOSA, 2013). Esses cenários reforçam a necessidade de se desenvolver heurísticas efincientes para o escalonamento de tarefas, pois a heterogeneidade do hardware impacta diretamente o *makespan*, o balanceamento de carga e o tempo de espera das tarefas.

Fora do ambiente acadêmico, casos recentes mostram *startups* explorando computação **heterogênea** em nuvem (e até fora de *data centers* tradicionais) para treinar LLMs. Reportagem da *WIRED* descreve a cooperação entre a Flower AI e a Vana para treinar o modelo *Collective-1* (cerca de 7B parâmetros) distribuindo o treinamento por centenas de máquinas com GPUs dispersas geograficamente (com gerações e capacidades distintas) utilizando ferramentas abertas para coordenar a execução (KNIGHT, 2025). Essa experiência reforça a tendência de aproveitar recursos heterogêneos e intermitentes, aproximando a prática industrial da realidade acadêmica (laboratórios com hardware de múltiplas gerações) e evidenciando a importância de heurísticas de escalonamento capazes de lidar com variações de desempenho, largura de banda e latência entre nós.

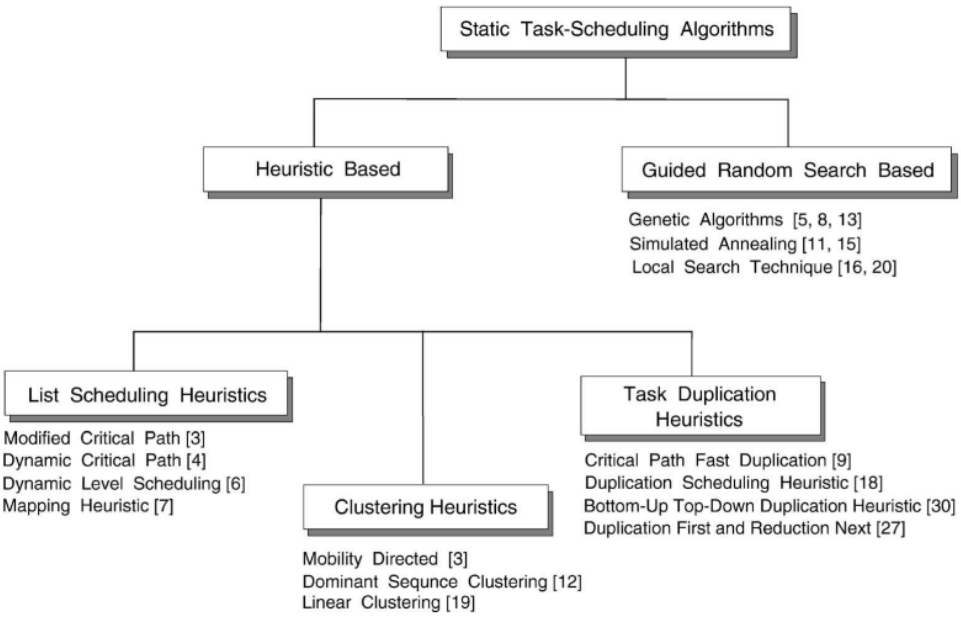
1.1 Contextualização do problema de escalonamento em sistemas heterogêneos

O escalonamento de aplicações paralelas em *clusters* heterogêneos é classicamente modelado por um grafo acíclico direcionado (DAG), no qual vértices representam tarefas (com tempos de execução distintos por processador) e arestas codificam dependências e custos médios de comunicação. O objetivo é minimizar o *makespan* (tempo de conclusão da última tarefa) respeitando precedências, variações de processamento e comunicação. Tal problema é *NP-completo*, o que motiva heurísticas de baixa complexidade em vez de métodos exatos (TOPCUOGLU; HARIRI; WU, 2002; ARABNEJAD; BARBOSA, 2013).

Seguindo a taxonomia consagrada, o problema é particionado em **escalonamento estático** (decisões em tempo de compilação, sem sobrecarga em execução) e **dinâmico** (decisões em tempo de execução, com sobrecarga adicional). No escopo estático, distinguem-se dois grandes grupos: *heuristic-based* e *guided random search-based*. O primeiro reúne soluções aproximadas de tempo polinomial; o segundo melhora a qualidade com iterações estocásticas adicionais (*genetic algorithms*, *simulated annealing*, busca local), porém a custo computacional mais elevado (ARABNEJAD; BARBOSA, 2013).

Dentro do grupo *heuristic-based*, três subfamílias são amplamente empregadas (Fi-

Figura 1 – Classificação de algoritmos *estáticos* de escalonamento: (i) *Heuristic-based* (subdivididos em *List Scheduling*, *Clustering* e *Task Duplication*) e (ii) *Guided Random Search-based* (e.g., AG, *simulated annealing*, busca local).



Fonte: Adaptado de Topcuoglu, Hariri e Wu (2002), Arabnejad e Barbosa (2013).

gura 1)

- **List Scheduling Heuristics** (*List-based*): atribuem uma prioridade a cada tarefa (fase de *priorização*) e, em seguida, escolhem um processador que minimiza uma função de custo (fase de seleção). Têm, em geral, complexidade quadrática no número de tarefas e produzem cronogramas eficientes. O HEFT (*Heterogeneous Earliest Finish Time*) tornou-se referência por conciliar *robustez* de resultados e complexidade $O(v^2p)$ (TOPCUOGLU; HARIRI; WU, 2002; ARABNEJAD; BARBOSA, 2013).
- **Clustering Heuristics**: agrupam tarefas (muitas vezes visando reduzir comunicação) para depois mapear os clusters a processadores. São tradicionais em sistemas homogêneos e apresentam limitações quando a heterogeneidade é elevada (ARABNEJAD; BARBOSA, 2013).
- **Task Duplication Heuristics**: minimizam comunicação duplicando tarefas em múltiplos processadores; embora gerem, em média, menores *makespans*, implicam complexidade cúbica e maior consumo de recursos/energia (ARABNEJAD; BARBOSA, 2013).

O presente trabalho concentra-se em **seis heurísticas *list-based*** selecionadas de forma cronológica para evidenciar a evolução das ideias e, posteriormente, compará-

las empiricamente em DAGs grandes que simulam o treinamento distribuído de LLMs em *clusters* heterogêneos: *DLS* (SIH; LEE, 1993; HAGRAS; JANEČEK, 2003), *HEFT* (TOPCUOGLU; HARIRI; WU, 2002), *HEFT-LA* (BITTENCOURT; SAKELLARIOU; MADEIRA, 2010), *PEFT* (ARABNEJAD; BARBOSA, 2013), *IHEFT* (ALEBRAHIM; AHMAD, 2016) e *IPEFT* (ZHOU et al., 2016). O PEFT introduz uma *Optimistic Cost Table* (OCT) para antecipar efeitos sobre os descendentes mantendo a complexidade de $O(v^2p)$ e, em média, reduzindo o *makespan* face ao HEFT (ARABNEJAD; BARBOSA, 2013); o IPEFT estende essa linha com uma *Pessimistic Cost Table* (PCT) e uma *Critical Node Cost Table* (CNCT), aumentando a frequência de melhores resultados e a robustez, sem elevar a ordem assintótica (ZHOU et al., 2016).

Por fim, alinhamos esta contextualização com trabalhos prévios nacionais que implementam e comparam heurísticas clássicas sob um conjunto comum de instâncias e métricas (e.g., *makespan*, *load balance*) como é feito em (COSTA, 2022), e estendemos aqui com a comparação de *novas heurísticas e métricas* relevantes ao treinamento de LLMs.

1.1.1 Modelagem do problema

Aplicações paralelas são comumente modeladas por um grafo acíclico direcionado (DAG), no qual vértices representam tarefas (com tempos de execução que variam por processador) e arestas codificam dependências e custos de comunicação médios entre dispositivos (TOPCUOGLU; HARIRI; WU, 2002; ARABNEJAD; BARBOSA, 2013). O objetivo clássico é mapear e ordenar as tarefas sobre um conjunto de processadores de modo a *minimizar o makespan*. Mesmo sob hipóteses simplificadoras, a literatura demonstra que o escalonamento de DAGs é **NP-completo** (i.e., intratável em geral), por consequência, a comunidade tem se concentrado em heurísticas de baixa complexidade que produzam bons cronogramas na prática (ARABNEJAD; BARBOSA, 2013; ZHOU et al., 2016).

Para ambientes heterogêneos, heurísticas *list-based* destacam-se por custo quadrático (em v tarefas) e boa qualidade: o **HEFT** (*Heterogeneous Earliest Finish Time*) prioriza tarefas pela métrica rank_u (comprimento do caminho restante até a saída, com custos médios de computação e comunicação) e seleciona o processador que minimiza o *Earliest Finish Time* (EFT), usando *insertion-based policy* para preencher *lacunas* ociosas (TOPCUOGLU; HARIRI; WU, 2002). Com complexidade $O(v^2 \cdot p)$ (v: tarefas; p: processadores), o HEFT é referência de robustez e *makespan* em comparativos clássicos de dezenas de heurísticas (ARABNEJAD; BARBOSA, 2013).

A partir do HEFT, diversas extensões foram propostas para melhor prever efeitos de *look-ahead* e de nós críticos sem aumentar a ordem assintótica. O **PEFT** (*Predict Earliest Finish Time*) introduz a *Optimistic Cost Table* (OCT) para ranqueamento e seleção, mantendo $O(v^2 \cdot p)$ e reportando melhorias de *makespan* (ARABNEJAD; BARBOSA,

2013). O **IPEFT** combina uma *Pessimistic Cost Table* (PCT) com uma *Critical-Node Cost Table* (CNCT), priorizando pais de nós críticos e preservando a mesma complexidade (ZHOU et al., 2016). Variantes *lookahead* do HEFT também aparecem na literatura (*HEFT-LA*) para antecipar o impacto da alocação corrente nos filhos da tarefa (BITTENCOURT; SAKELLARIOU; MADEIRA, 2010).

1.2 Motivação deste trabalho.

No contexto do treinamento distribuído de LLMs, a heterogeneidade se manifesta na diversidade de gerações de GPUs, topologias de rede (NVLink/PCIe/Ethernet), memórias e *batching/pipeline* paralelos. Medidas como **makespan**, **balanceamento de carga** e **tempo de espera** (*waiting time*) relacionam-se diretamente à vazão do treino e ao custo por experimento; já a **eficiência energética** desponta como métrica de primeira classe pela relevância econômica e ambiental (STRUBELL; GANESH; MCCALLUM, 2019; SCHWARTZ et al., 2020; PATTERSON et al., 2021; HENDERSON et al., 2020; LACOSTE et al., 2019).

1.2.1 Metodologia

Nesta monografia, adotamos uma abordagem *reprodutível* e *paper-faithful* para a comparação de heurísticas clássicas de *list scheduling* em ambientes heterogêneos. Partimos das referências-base de cada algoritmo e implementamos a "versão clássica" a partir do pseudocódigo e da descrição textual originais, preservando *estritamente* os objetivos e os parâmetros definidos nos artigos-fonte, sem introduzir variações que pudessem favorecer um método específico.

i) **DLS** (*Dynamic Level Scheduling*) (SIH; LEE, 1993);

ii) **HEFT** (*Heterogeneous Earliest Finish Time*) (TOPCUOGLU; HARIRI; WU, 2002);

iii) **HEFT-LA** (variante com *look-ahead*) (BITTENCOURT; SAKELLARIOU; MADEIRA, 2010);

iv) **PEFT** (*Predict Earliest Finish Time*, com *Optimistic Cost Table*) (ARABNEJAD; BARBOSA, 2013);

v) **IPEFT** (*Improved PEFT*, com *Pessimistic Cost Table* e *Critical Node Cost Table*) (ZHOU et al., 2016);

vi) **IHEFT** (*Improved HEFT*) (ALEBRAHIM; AHMAD, 2016).

Para garantir um processo justo e fidedigno, mantivemos: (i) o mesmo modelo de comunicação (custos por aresta do DAG, com banda/latência médias quando aplicável), (ii) a mesma política de inserção de lacunas quando definida pelo algoritmo (e.g., HEFT, PEFT e IPEFT), (iii) os mesmos critérios de priorização (e.g., *upward rank*, OCT, PCT e CNCT) e (iv) as mesmas regras de desempate descritas nas fontes. Todas as heurísticas foram executadas sobre o mesmo conjunto de instâncias (*benchmarks* sintéticos gerados e grafos canônicos), e as métricas relatadas incluem *makespan*, *load balance*, *waiting time* e *custo de consumo energético*, conforme discutido na Seção 1.1.

Os detalhes de implementação (pseudocódigos adotados, estruturas de dados, cálculo de *ranks* e seleção de processador, bem como particularidades de cada heurística) serão apresentados no Capítulo *Desenvolvimento*, com a devida rastreabilidade para as fórmulas e passos descritos nas referências citadas.

1.3 Objetivo e contribuições

Seguindo a linha metodológica de Costa (2022), realizamos uma comparação *lado a lado*, no *mesmo* ambiente heterogêneo, de seis algoritmos clássicos: DLS, HEFT, HEFT-LA, PEFT, IHEFT e IPEFT. Incorporamos métricas adicionais — *waiting time* e *custo energético* — e adotamos parametrizações de computação e comunicação alinhadas a cenários de treino de LLMs. As métricas avaliadas são: **makespan**, **balanceamento de carga**, **tempo de espera** (*waiting time*) e **eficiência energética** (esta última apenas reportada). Além de consolidar resultados sob configuração reprodutível, discutimos hipóteses de modelagem (p. ex., política de inserção, comunicação média, não preempção) e sua adequação a *clusters* heterogêneos modernos.

O restante deste documento está organizado da seguinte forma:

- **Capítulo 2 – Fundamentação Teórica**: formaliza a modelagem por DAGs, define notação, custos de computação e comunicação e métricas de avaliação; incorpora a *revisão bibliográfica* das heurísticas de *list scheduling* (DLS, HEFT, HEFT-LA, PEFT, IHEFT, IPEFT) e discute hipóteses de modelo.
- **Capítulo 3 – Desenvolvimento:** detalha os algoritmos analisados (formulação, pseudocódigos, hipóteses e complexidade), as decisões de implementação e a **geração sintética de grafos de tarefas** utilizada posteriormente nos experimentos.
- **Capítulo 4 – Experimentos:** descreve os *workloads* (DAGs de teste), a plataforma e a metodologia, os parâmetros e as métricas, e apresenta/discute os resultados quantitativos comparando DLS, HEFT, HEFT-LA, PEFT, IHEFT e IPEFT.

- **Capítulo 5 – Conclusão:** sintetiza as contribuições, limitações, implicações para treinamento de LLMs e aponta direções para trabalhos futuros.
- **Apêndices e Anexos:** reúnem material de reprodutibilidade (scripts, configs e instruções), definições e resultados complementares, trabalhos correlatos não incluídos na avaliação principal e o manifesto de versão/licenças.

1.4 Implementação e Ambiente Experimental

1.4.1 Implementação

Este projeto provê implementações *paper-faithful* em **Python** das seis heurísticas clássicas de *list scheduling* (**DLS**, **HEFT**, **HEFT-LA**, **PEFT**, **IHEFT**, **IPEFT**), em uma base reprodutível que padroniza formato de entrada/saída e cálculo de métricas, assegurando que diferenças observadas decorram da lógica de escalonamento e não de instrumentação (MONTEIRO, 2025). O repositório é licenciado em **MIT** e corresponde ao **TCC da UFU/FACOM**.

Ponto de entrada: `compare_same_dataset.py` (script principal). Ele carrega um conjunto de *CSVs* (conectividade do DAG, tempos de execução, largura de banda e, opcionalmente, matriz de potência) e executa uma ou mais heurísticas, imprimindo métricas em JSON (1 linha por algoritmo) e, se desejado, salvando artefatos como Gantt, DAG e gráfico de métricas.

Argumentos essenciais (resumo): `-dag`, `-exec`, `-bw`, `-power` (opcional), `-algos`, `-out_dir`, `-save_dag`, `-plot_metrics`.

Dependências: instalação via `requirements.txt`; caso mínimo manual: `numpy`, `networkx`, `matplotlib`, `pydot`. O **Graphviz** é opcional (melhor *layout* do DAG); sem ele, um *fallback layout* é usado automaticamente.

Fluxo típico:

1. Escolher um conjunto de CSVs (cenário/tamanho do DAG e número de processadores).
2. Rodar `compare_same_dataset.py` para as variantes (tarefas, processadores).
3. (Opc.) Agregar com `aggregate_all_results.py` e visualizar tendências com `plot_aggregate_`

Exemplos de execução:

```
# Métricas no console (exemplo mínimo)
python compare_same_dataset.py \
  --dag  graphs/canonicalgraph_task_connectivity.csv \
  --exec graphs/canonicalgraph_task_exe_time.csv \
  --bw   graphs/canonicalgraph_resource_BW.csv \
  --algos HEFT,PEFT

# Execução completa com artefatos (cenário 1: 256 tarefas, 8 PEs)
python compare_same_dataset.py \
  --dag   graphs/Scenario_1/peft256_8proc_task_connectivity.csv \
  --exec  graphs/Scenario_1/peft256_8proc_task_exe_time.csv \
  --bw    graphs/Scenario_1/peft256_8proc_resource_BW.csv \
  --power graphs/Scenario_1/peft256_8proc_task_power.csv \
  --algos DLS,HEFT,HEFT-LA,PEFT,IHEFT,IPEFT \
  --out_dir outputs/scenario1_256x8_labels \
  --save_dag --plot_metrics
```

1.4.1.0.1 Detalhes complementares

Procedimentos completos de reprodução (estrutura de pastas, versões e *hash* do commit, preparo de ambiente, variações de linha de comando e checklist de auditoria) estão documentados no Apêndice A.

1.4.2 Modelo de Entrada e Métricas

Cada instância é especificada por 3–4 matrizes CSV: (1) conectividade (tarefas×tarefas; peso de aresta = volume de dados); (2) tempos de execução (tarefas×processadores); (3) largura de banda (processadores×processadores); (4) potência (opcional; tarefas×processadores). As métricas incluem *makespan*, utilização média, tempo médio de espera e energia (pós-cálculo usando duração e potência por tarefa).

1.4.3 Ambiente de Execução e Hardware

As execuções consistem em **simulação determinística** a partir de CSVs; não há uso de kernels de GPU nem dependência de APIs específicas de *runtime* de aceleradores. Assim, o *hardware* físico impacta apenas o *tempo de simulação*, mas não altera as métricas para um mesmo dataset e semente.

Para transparência, registramos a seguinte configuração de referência utilizada para gerar os resultados desta monografia (substitua pelos seus dados exatos, se desejar):

- **SO**: Windows 11 Pro (build 22631) ou Ubuntu 22.04 LTS;
- **Python**: 3.10+; **Bibliotecas**: `numpy`, `networkx`, `matplotlib`, `pydot`;
- **CPU**: classe 8 c/16 t (e.g., Ryzen 7/Intel i7); **RAM**: $\geq$16 GB; **Armazenamento**: SSD NVMe;
- **GPU(s)**: *não requerida(s)* para a simulação (apenas modeladas como PEs nos CSVs);
- **Reprodutibilidade**: `requirements.txt`, *seeds* fixas, scripts de agregação/plotagem.

1.4.4 Boas Práticas de Performance

Para DAGs muito grandes ($\geq$2048 tarefas), recomenda-se omitir `-save_dag` (gera o layout do grafo) e postergar `-plot_metrics` para a passada final; ao iterar rapidamente, restringir a lista de algoritmos (e.g., `-algos HEFT,PEFT`) e começar por cenários menores (256×8) acelera o ciclo de desenvolvimento e análise.

2 Fundamentação Teórica

Estabelecemos a notação, os modelos e as métricas usados ao longo do trabalho, separando claramente o que é *entrada do problema* (o que o algoritmo recebe) do que é *saída de avaliação* (o que medimos após a execução).

O escalonamento de tarefas em sistemas heterogêneos é, no caso geral (DAG $\rightarrow$ máquinas distintas), um problema *NP-completo* (TOPCUOGLU; HARIRI; WU, 2002). Assim, este trabalho adota heurísticas de *list scheduling* clássicas, modelando aplicações como DAGs e ambientes com custos de computação e comunicação.

2.1 Modelo de Tarefas (DAG)

Esta seção formaliza a aplicação como um grafo acíclico direcionado, fixando nomenclatura para tarefas, dependências e nós artificiais de entrada/saída – base para caminhos críticos e precedências.

Representamos a aplicação por um grafo acíclico dirigido $G = (V, E)$, em que cada vértice $n_i \in V$ é uma tarefa e cada aresta $(n_m, n_i) \in E$ codifica precedência/fluxo de dados. Definimos

$$\text{pred}(i) = \{\, n_m \mid (n_m, n_i) \in E \,\}, \qquad \text{succ}(i) = \{\, n_j \mid (n_i, n_j) \in E \,\}.$$

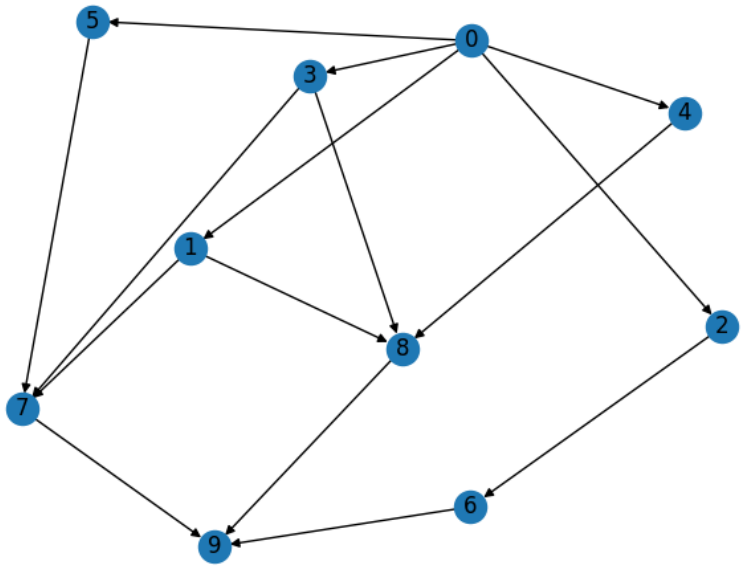
Nós artificiais *entry*/*exit* podem ser usados para unificar origens e sumidouros (TOPCUOGLU; HARIRI; WU, 2002).

A Figura 2 traz um exemplo de DAG com nove tarefas, tendo como origem o nó 0 e sumidouro o nó 9.

2.1.1 Atomicidade e não preempção

Adotamos a hipótese de *atomicidade* das tarefas: cada tarefa é indivisível (*non-preemptive*) e, uma vez iniciada em um processador p_j, executa até a conclusão no *mesmo* processador; não há migração ou interrupção durante sua execução. Essa suposição é padrão nos modelos de escalonamento estático por lista em plataformas heterogêneas – como nos algoritmos HEFT, PEFT e IPEFT – e é compatível com a política de inserção (*insertion-based*) típica dessas heurísticas, que aloca cada tarefa em um intervalo ocioso contíguo na linha do tempo do processador, respeitando janelas sem conflitos e as precedências (TOPCUOGLU; HARIRI; WU, 2002; ARABNEJAD; BARBOSA, 2013; ZHOU et al., 2016). Consequentemente, a comunicação associada a uma aresta (n_i, n_k) somente ocorre após o término de n_i; se n_i e n_k forem mapeadas para o mesmo processador, o

Figura 2 – Exemplo de modelagem de aplicação como um DAG de tarefas (arestas indicam precedências e fluxo de dados).



custo de comunicação é desprezível (assumido como zero), ao passo que, em processadores distintos, considera-se latência e banda da interconexão no cálculo do tempo de transferência. Esse modelo – amplamente adotado na literatura – preserva a atomicidade e reflete como os DAGs são de fato tratados por heurísticas de *list scheduling*.

2.2 Ambiente Heterogêneo e Custos

Descrevemos os recursos (CPUs e GPUs) e como modelamos tempos de computação e comunicação entre dispositivos, compondo as matrizes/tensores de custo usadas pelos algoritmos.

Seja $\mathcal{P} = \{p_1, \ldots, p_P\}$ o conjunto de processadores (CPUs/GPUs). O **tempo de computação** da tarefa n_i em p_j é $w_i^j > 0$ (matriz $W = [w_i^j]$). Para $p_a \neq p_b$, o **custo de comunicação** na aresta (n_m, n_i) é

$$c_{m,i}^{a,b} \;=\; \lambda_{a,b} \;+\; \frac{d_{m,i}}{B_{a,b}},$$

com latência $\lambda_{a,b}$, largura de banda $B_{a,b}$ e volume de dados $d_{m,i}$ (para $a = b$, $c_{m,i}^{a,a} = 0$).

2.3 Cronograma e Tempos (EST e EFT)

Definimos o que é um cronograma (mapeamento tarefa→processador) e os tempos relevantes (início/término efetivos e estimados), além da política de inserção em lacunas típica nas heurísticas por lista.

Um escalonamento consiste no mapeamento $\sigma : V \to \mathcal{P}$ e nos tempos efetivos $\mathrm{AST}(n_i)$ e $\mathrm{AFT}(n_i)$, respeitando

$$\mathrm{AST}(n_i) \;\geq\; \max_{k \in \mathrm{pred}(i)} \Big(\mathrm{AFT}(k) + \mathbf{1}[\sigma(k) \neq \sigma(i)]\, c_{k,i}^{\sigma(k),\sigma(i)} \Big).$$

Com inserção em lacunas por processador, definimos, dado o estado parcial e a disponibilidade $\mathrm{avail}(p)$:

$$\mathrm{EST}(i,p) = \max\Big(\mathrm{avail}(p),\; \max_{k \in \mathrm{pred}(i)} (\mathrm{AFT}(k) + c_{k,i}^{\sigma(k),p}) \Big), \qquad \mathrm{EFT}(i,p) = \mathrm{EST}(i,p) + w_i^p.$$

2.4 Parâmetros de Entrada

Listamos os dados fornecidos aos algoritmos antes da execução; energia aparece apenas como parâmetro para mensuração posterior, não influenciando decisões nas heurísticas clássicas.

1. Número de processadores P e tarefas $|V|$.
2. Matriz de computação $W = [w_i^j]$.
3. Parâmetros de comunicação $(\lambda_{a,b}, B_{a,b}, d_{m,i})$ que induzem $C = [c_{m,i}^{a,b}]$.
4. **Coeficiente energético por processador** $\eta = (\eta_1, \ldots, \eta_P)$, usado apenas para *medir* energia (ver Métricas).

2.5 Métricas de Avaliação

Definimos as métricas usadas para comparar algoritmos após o escalonamento: desempenho (makespan), uso de recursos (balanceamento), latências internas (tempo de espera) e impacto energético (custo de energia).

2.5.1 Makespan

Tempo total até a conclusão da última tarefa – sintetiza, em um único número, as escolhas de priorização, alocação e comunicação.

$$\mathrm{MS} = \max_{n_i \in V} \mathrm{AFT}(n_i).$$

2.5.2 Balanceamento de carga (*Load Balancing*)

Mede o quão uniformemente os processadores ficaram ocupados ao longo da execução; valores próximos de 1 indicam melhor aproveitamento médio. Para cada p_j,

$$T_j^{(\mathrm{ocup})} = \sum_{n_i:\, \sigma(n_i)=p_j} \Big(\mathrm{AFT}(n_i) - \mathrm{AST}(n_i) \Big), \quad \bar{U} = \frac{\frac{1}{P} \sum_{j=1}^{P} T_j^{(\mathrm{ocup})}}{\mathrm{MS}} \in [0,1].$$

2.5.3 Tempo médio de espera (*Waiting Time*)

Captura o atraso antes do início de cada tarefa, refletindo dependências (pais) e contenção de recursos (filas por processador).

$$\mathrm{WT}_i = \mathrm{AST}(n_i), \qquad \overline{\mathrm{WT}} = \frac{1}{|V|}\sum_i \mathrm{WT}_i.$$

2.5.4 Custo de energia (pós-execução)

Estima a energia proporcional ao tempo de processamento por processador; é uma *métrica derivada* do cronograma e não altera a execução das heurísticas clássicas. Adotamos o princípio "*mais processamento* ⇒ *maior custo energético*". Modelamos uma matriz energética $\mathcal{E} = [E_i^j]$ com

$$E_i^j = \eta_j \, w_i^j,$$

onde η_j é um coeficiente (energia por unidade de tempo) do processador p_j. Dado o cronograma final,

$$E_{\text{total}} = \sum_{n_i \in V} E_i^{\sigma(n_i)} = \sum_{n_i \in V} \eta_{\sigma(n_i)} \, w_i^{\sigma(n_i)}.$$

Observação: essa métrica é *derivada* do cronograma e *não* altera a execução das heurísticas avaliadas. Mantemos a implementação fiel à metodologia original de cada artigo clássico, que não recebe energia como parâmetro de decisão.

2.6 Revisão Bibliográfica

Nesta seção, sintetizamos a evolução das principais heurísticas de escalonamento estático por lista (*list scheduling*) para sistemas heterogêneos, tendo como eixo comum a modelagem das aplicações em DAGs e o objetivo de minimizar o *makespan* sob custos distintos de computação e comunicação. Partimos do *Dynamic Level Scheduling* (DLS) (SIH; LEE, 1993), que introduz o nível dinâmico e uma política gulosa sensível à disponibilidade dos processadores; avançamos para o *Heterogeneous Earliest Finish Time* (HEFT) (TOPCUOGLU; HARIRI; WU, 2002), que consolida a priorização por *ranku* e a seleção por *EFT* com inserção em lacunas; examinamos o *HEFT-Lookahead* (BITTENCOURT; SAKELLARIOU; MADEIRA, 2010), que antecipa o impacto da alocação nos sucessores; discutimos a família preditiva *PEFT* (ARABNEJAD; BARBOSA, 2013), baseada na *Optimistic Cost Table* (OCT); e suas variantes sensíveis à criticidade, *IHEFT* (ALEBRAHIM; AHMAD, 2016) e *IPEFT* (ZHOU et al., 2016), que incorporam tabelas pessimistas e de nós críticos para refinar a previsão sem aumentar a complexidade assintótica de HEFT/PEFT.

Ao longo da revisão, enfatizamos hipóteses de modelo (rede totalmente conectada, comunicação sem contenção, execução não preemptiva), complexidade temporal, e métricas de comparação (como *makespan*, balanceamento de carga e *waiting time*), preparando

o terreno para a análise experimental que conecta essas heurísticas a cenários inspirados em treinamento distribuído de LLMs.

O *Dynamic Level Scheduling* (DLS), proposto por Sih e Lee (1993), introduziu a métrica de *nível dinâmico* (DL) – diferença entre um nível estático e o menor tempo estimado de início – para priorização e seleção gulosa em grafos acíclicos direcionados (DAGs) executados sobre recursos heterogêneos. Apesar do custo computacional mais elevado quando comparado a heurísticas modernas, o DLS consolidou-se como *baseline* clássico em estudos comparativos.

O *Heterogeneous Earliest Finish Time* (HEFT), de Topcuoglu, Hariri e Wu (2002), tornou-se referência por combinar: (i) priorização por *ranku* (caminho crítico médio a partir da tarefa até a saída) e (ii) seleção do processador pelo menor *Estimated Finish Time* (EFT) com política de inserção em lacunas (*insertion-based*). Sua simplicidade e qualidade de solução explicam a ampla adoção do HEFT como *state-of-practice* em heterogeneidade.

Para mitigar decisões gulosas do HEFT, Bittencourt, Sakellariou e Madeira (2010) propuseram uma variante com *lookahead*, que estima o impacto da alocação atual nos sucessores (filhos) antes de decidir. Os resultados mostram reduções de *makespan* relevantes em cenários com comunicação dominante, a custo de maior complexidade temporal.

O *Predict Earliest Finish Time* (PEFT), proposto por Arabnejad e Barbosa (2013), mantém a mesma ordem de complexidade do HEFT, mas introduz a *Optimistic Cost Table* (OCT) e o *Optimistic EFT* (OEFT) para "prever" custos até a saída do grafo no ato da seleção do processador, aumentando a qualidade do escalonamento de forma sistemática, sobretudo em DAGs maiores.

O *Improved HEFT* (IHEFT) especializa o HEFT para grafos com caminhos críticos dominantes (ALEBRAHIM; AHMAD, 2016). A ideia central é dar preferência a tarefas críticas – identificadas via AEST/ALST ou estruturas correlatas – e preservar afinidades de comunicação, o que reduz ociosidade e conflitos de comunicação. Em contrapartida, há um aumento de custo computacional em relação ao HEFT original.

Finalmente, o *Improved Predict Earliest Finish Time* (IPEFT) estende o PEFT substituindo a única OCT por duas tabelas (ZHOU et al., 2016): uma *pessimistic cost table* (PCT) e uma *critical-node cost table* (CNCT). A priorização combina o custo "até a saída" mais o custo médio de computação, e a seleção de processador minimiza um EFT penalizado pela CNCT (exceto em casos de pais críticos), melhorando a robustez da previsão e mantendo a complexidade assintótica do HEFT/PEFT.

2.7 Trabalhos relacionados

Canon et al. (2008) apresentam uma avaliação comparativa da *robustez* de 20 heurísticas estáticas de escalonamento de DAGs em ambientes heterogêneos, originalmente voltadas à minimização do *makespan*. A metodologia mede robustez pela variação do makespan sob perturbações controladas nos tempos de computação/comunicação e investiga se reordenações dinâmicas locais nos processadores melhoram a robustez (resultado geral: respeitar a ordem estática tende a ser preferível). Dentre as heurísticas avaliadas, o **HEFT** figura entre as que conciliam bom makespan com robustez (CANON et al., 2008).

Este trabalho difere em três frentes. (i) *Escopo algorítmico*: comparamos um subconjunto moderno de heurísticas por lista — **DLS**, **HEFT**, **HEFT-LA**, **PEFT**, **IHEFT** e **IPEFT** — do qual apenas o **HEFT** aparece em (CANON et al., 2008) (HEFT-LA/PEFT/IHEFT/IPEFT são posteriores). (ii) *Métricas*: além de **makespan** (e SLR), incluímos **balanceamento de carga**, **tempo de espera** (*waiting time*) e **eficiência/custo energético**. Em suma, enquanto (CANON et al., 2008) estabelece um panorama amplo de robustez para 20 heurísticas clássicas, este trabalho aprofunda a comparação de heurísticas por lista recentes sob métricas e custos específicos e com exemplos de aplicações ao treinamento de modelos de aprendizado de máquina.

2.8 Considerações Finais

Com a ascensão de treinamentos de LLMs sobre clusters heterogêneos de GPUs, surgem arquiteturas e sistemas que explicitamente exploram heterogeneidade e comunicação para reduzir *makespan* e aumentar utilização. Exemplos recentes incluem abordagens que acomodam variabilidade de GPU, largura de banda e interconexão, com divisão hierárquica de fases e alocação consciente de comunicação (YAN et al., 2025). Tais sistemas evidenciam que princípios dos algoritmos HEFT, PEFT e IPEFT podem ser adaptados para pipelines de treinamento distribuído de LLMs, onde dependências (*forward/backward*, sincronização, *all-reduce*) se mapeiam naturalmente em DAGs.

Em síntese, a literatura evolui de heurísticas clássicas (DLS) a estratégias "previsoras" (PEFT e IPEFT) e variantes sensíveis a criticidade (IHEFT), oferecendo um leque de trade-offs entre custo computacional e qualidade de solução. Este trabalho usa esse arcabouço como base para comparar, sob métricas como *makespan*, balanceamento de carga e *waiting time*, o desempenho desses algoritmos em cenários ajustados ao treinamento de LLMs em ambientes heterogêneos.

3 Desenvolvimento

O treinamento distribuído de modelos de linguagem de grande porte (LLMs) em clusters heterogêneos envolve a divisão do trabalho de treinamento (forward, backward e otimização) em múltiplas tarefas interdependentes, que devem ser alocadas a recursos de hardware distintos (GPUs/CPUs com diferentes capacidades). Escalonar eficientemente essas tarefas é crucial para minimizar o tempo total de treinamento (makespan) e maximizar o uso dos recursos disponíveis. Entretanto, encontrar o escalonamento ótimo é um problema NP-completo, especialmente em ambientes heterogêneos onde cada recurso tem desempenho distinto. Por isso, diversas heurísticas de escalonamento têm sido propostas. Esta seção foca em seis algoritmos clássicos de escalonamento estático de *workflows* (DAGs de tarefas) em sistemas heterogêneos – DLS, HEFT, HEFT-LA(Lookahead), PEFT IHEFT e PEFT – discutindo seu funcionamento, complexidade e aplicabilidade prática no contexto de treinamento distribuído de LLMs. Em seguida, comparam-se as características de desempenho desses algoritmos considerando as seguintes métricas: **makespan, balanceamento de carga, tempo de espera (waiting time) e eficiência energética**. Referências a trabalhos recentes e casos de uso ilustram a aplicação desses algoritmos em contextos similares (e.g. clusters de GPUs heterogêneas).

3.1 Algoritmos Selecionados

Neste trabalho foram selecionados seis algoritmos para a análise comparativa – *Dynamic Level Scheduling* (DLS), *Heterogeneous Earliest Finish Time* (HEFT), *Lookahead Scheduling* (HEFT-LA), *Predict Earliest Finish Time* (PEFT), *Improved Heterogeneous Earliest Finish Time* (IHEFT) e *Improved Predict Earliest Finish Time* (IPEFT) – devido às suas características complementares e à relevância consolidada na literatura de escalonamento em sistemas heterogêneos (SIH; LEE, 1993; TOPCUOGLU; HARIRI; WU, 2002; BITTENCOURT; SAKELLARIOU; MADEIRA, 2010; ARABNEJAD; BARBOSA, 2013; ZHOU et al., 2016).

O **DLS** foi escolhido por ser uma heurística clássica que ajusta dinamicamente a prioridade das tarefas a cada decisão de alocação, servindo como *baseline* histórico para comparar soluções mais recentes (SIH; LEE, 1993). O **HEFT** destaca-se pela ampla aceitação na comunidade, pela simplicidade conceitual e pela boa relação entre qualidade do escalonamento e complexidade, combinando uma fase de priorização por *rank* (via caminho ascendente) e uma fase de seleção de processador pela minimização do *Earliest Finish Time* com política de inserção (TOPCUOGLU; HARIRI; WU, 2002).

O **HEFT-LA** (*Lookahead*) é uma variante que antecipa impactos de decisões lo-

cais sobre tarefas descendentes, reduzindo escolhas localmente ótimas porém globalmente subótimas; ele avalia explicitamente combinações para os filhos durante a seleção de processador (BITTENCOURT; SAKELLARIOU; MADEIRA, 2010). O **PEFT** introduz a *Optimistic Cost Table* (OCT) para incorporar um mecanismo de *lookahead* leve durante a priorização e a seleção, mantendo a mesma ordem de complexidade assintótica do HEFT e apresentando melhorias consistentes de *makespan* em diversos cenários (ARABNEJAD; BARBOSA, 2013).

O **IHEFT** foi incluído por representar uma evolução direta do HEFT: trata-se de uma família de variações que aprimoram (i) o esquema de priorização/rankeamento e (ii) a regra de seleção/inserção no processador, visando reduzir atrasos em caminhos críticos e melhorar a ocupação dos recursos em ambientes fortemente heterogêneos – típicos do treinamento distribuído de LLMs. Em essência, o IHEFT mantém o espírito do HEFT (lista + inserção), mas reequilibra o peso dado ao caminho crítico e às lacunas de *schedule*, frequentemente obtendo cronogramas mais curtos sem elevar a complexidade; adotamos aqui a implementação clássica descrita nas referências que apresenta uma nova proposta do algoritmo HEFT desenvolvido em (TOPCUOGLU; HARIRI; WU, 2002), assim como fez (COSTA, 2022), neste trabalho, nomeamos a implementação de (ALEBRAHIM; AHMAD, 2016) de IHEFT que significa “Improved HEFT”, assim como Zhou et al. (2016) sugeriu como uma melhoria do trabalho de Arabnejad e Barbosa (2013).

Por fim, o **IPEFT** estende o PEFT substituindo a OCT por duas tabelas de custo: a *Pessimistic Cost Table* (PCT) e a *Critical Node Cost Table* (CNCT). A PCT confere um viés conservador às estimativas futuras enquanto a CNCT enfatiza nós críticos e seus pais, priorizando caminhos de maior impacto no *makespan*. Esse desenho aumenta a robustez (menor variância de resultados) e a frequência de melhores escalonamentos em relação ao HEFT/PEFT, preservando a complexidade assintótica de *list scheduling* (ZHOU et al., 2016).

Em conjunto, os seis algoritmos (DLS, HEFT, HEFT-LA, PEFT, IHEFT e IPEFT) abrange desde o *baseline* dinâmico clássico até heurísticas modernas com *lookahead* e atenção explícita a nós críticos. Isso nos permite uma comparação abrangente sob métricas relevantes ao contexto de LLMs (como *makespan*, *SLR*, *load balance*, *waiting time* e custo energético), em ambientes com CPUs/GPUs de diferentes gerações e interconexões. As próximas subseções detalham a formulação, o pseudocódigo e as decisões de implementação de cada algoritmo, mantendo rastreabilidade com as referências primárias.

3.1.1 Dynamic Level Scheduling (DLS)

O *Dynamic Level Scheduling* (DLS) (SIH; LEE, 1993) é uma heurística de *list scheduling* para sistemas heterogêneos que, a cada iteração, escolhe o par tarefa–processador que maximiza o **nível dinâmico** (*dynamic level*, DL). O DL de uma tarefa t_i no proces-

sador p_j é definido como a diferença entre o **nível estático** (*static level*, SL) da tarefa e o seu **tempo mais cedo de início** (*Earliest Start Time*, EST) no processador considerado, i.e., $DL(t_i, p_j) = SL(t_i) - EST(t_i, p_j)$. Com isso, o DLS pondera a "importância global" (via SL) com o estado corrente de disponibilidade dos processadores (via EST), selecionando sempre o par com maior DL.

Em termos de comportamento, o DLS tende a priorizar tarefas de *alto* SL no início do escalonamento e, à medida que o grafo vai "esvaziando", privilegia tarefas com menor EST, aproximando-se do critério local de início mais cedo. Diferentemente de variantes modernas (e.g., HEFT), o DLS clássico realiza a seleção de processadores com base em EST e *não* emprega política de inserção de lacunas (*insertion-based policy*) para preencher *idle slots*, o que pode levar à perda de oportunidades de reduzir o *makespan* em cenários com buracos de agenda entre tarefas.

Complexidade. Seja $v{=}|V|$ o número de tarefas do DAG e $p{=}|P|$ o número de processadores. A complexidade assintótica do DLS, na versão para interconexão totalmente conectada, é $\mathcal{O}(v^3 \cdot p)$.

Equações basais.

Nível estático (SL), como maior caminho até o *exit* com custos médios:

$$\mathrm{SL}(i) = \overline{w}_i + \max_{j \in succ(i)} \Big(\overline{c}_{i,j} + \mathrm{SL}(j) \Big).$$

No estado t, nível dinâmico de i em p:

$$\mathrm{DL}(i, p, t) = \mathrm{SL}(i) - \mathrm{EST}(i, p, t),$$

onde $\mathrm{EST}(i, p, t)$ é calculado como no HEFT (com inserção), usando disponibilidades e AFT correntes.

O Algoritmo 1 apresenta o pseudocódigo do DLS, adaptado de Hagras e Janeček (2003).

Algoritmo 1 Dynamic Level Scheduling

Computar $\mathrm{SL}(i)$ para todas as tarefas
$R \leftarrow$ tarefas prontas; $U \leftarrow$ tarefas não alocadas
enquanto $U \neq \emptyset$ **faça**
    para cada $i \in R$ e $p \in P$ **faça**
        Calcular $\mathrm{EST}(i, p, t)$; $\mathrm{DL}(i, p, t) = \mathrm{SL}(i) - \mathrm{EST}(i, p, t)$
    Escolher $(i^*, p^*) = \arg\max_{i \in R,\, p \in P} \mathrm{DL}(i, p, t)$
    Alocar i^* em p^* (primeira lacuna viável); atualizar $\mathrm{AFT}(i^*)$, t e disponibilidades
    Atualizar R (inserir novas tarefas liberadas); remover i^* de U

Do ponto de vista de complexidade, o DLS realiza, a cada inserção de tarefa, uma busca pelo melhor processador considerando o DL de cada tarefa pronta. Sua complexidade de tempo é relativamente alta – em geral $O(p \cdot v^3)$, sendo v o número de

tarefas e p o número de processadores (HAGRAS; JANEČEK, 2003). Versões posteriores e generalizadas do DLS procuram melhorar essa complexidade, mas em comparação com algoritmos mais simples, o DLS pode se tornar computacionalmente custoso para grafos muito grandes. Ainda assim, o DLS foi um dos primeiros algoritmos capazes de lidar com heterogeneidade de forma eficaz, e serve de base para várias outras heurísticas. Ele não realiza duplicação de tarefas e busca minimizar explicitamente o makespan. Em estudos comparativos, o DLS apresentou desempenho competitivo, embora tenha sido superado por algoritmos mais recentes em muitos cenários (BASKIYAR; DICKINSON, 2005). Por exemplo, (TOPCUOGLU; HARIRI; WU, 2002) reportaram que o HEFT (discutido adiante) produz escalonamentos com tempo de conclusão menor que o DLS em média (BASKIYAR; DICKINSON, 2005).

No contexto de treinamento de LLMs distribuído, o DLS poderia ser aplicado modelando as operações do treinamento (por exemplo, computação de *forward*/*backward* de diferentes camadas do modelo, sincronização de gradientes, etc.) como um grafo acíclico de tarefas. Seu critério de seleção (DL) levaria em conta tanto a criticidade de certas etapas (e.g., atualizar parâmetros principais no otimizador) quanto a disponibilidade imediata de cada GPU. Entretanto, devido à sua alta complexidade, aplicar DLS diretamente em um grafo muito fino (por exemplo, cada operação de rede neural como um nó) seria inviável. Uma possibilidade prática é usar DLS em um nível mais *coarse-grained* – por exemplo, para escalonar blocos inteiros de camadas ou etapas macro do pipeline de treinamento entre GPUs heterogêneas.

3.1.2 Heterogeneous Earliest Finish Time (HEFT)

O *Heterogeneous Earliest Finish Time* (HEFT) (TOPCUOGLU; HARIRI; WU, 2002) é uma das heurísticas de *list scheduling* mais influentes para ambientes heterogêneos. Ele segue duas fases: (i) **priorização de tarefas** por *rank ascendente* e (ii) **seleção de processador** por *menor tempo de término* (*Earliest Finish Time*, EFT) com *política de inserção*.

Fase 1 — Prioridade (rank ascendente). Para cada tarefa n_i, define-se o *rank* ascendente como

$$\mathrm{rank}_u(n_i) \;=\; \overline{w_i} \;+\; \max_{n_j \in \mathrm{succ}(n_i)} \Big(\overline{c_{i,j}} + \mathrm{rank}_u(n_j) \Big),$$

onde $\overline{w_i}$ é o custo médio de computação de n_i nos processadores e $\overline{c_{i,j}}$ é o custo médio de comunicação na aresta (i, j). As tarefas são ordenadas por rank_u em ordem decrescente (TOPCUOGLU; HARIRI; WU, 2002; BASKIYAR; DICKINSON, 2005; SANDOKJI; EASSA, 2019).

Fase 2 — Alocação (seleção de processador). Na ordem de prioridade, para

cada tarefa n_i avaliam-se todos os processadores p_j e calcula-se o *EFT*:

$$\mathrm{EFT}(n_i, p_j) \;=\; \mathrm{EST}(n_i, p_j) + w_{i,j},$$

em que $w_{i,j}$ é o tempo de computação de n_i em p_j e $\mathrm{EST}(n_i, p_j)$ é o *Earliest Start Time*, dado pelo máximo entre (a) o instante disponível em p_j (considerando **política de inserção** para ocupar lacunas) e (b) o tempo em que todos os predecessores de n_i já terminaram e seus dados chegaram a p_j (considerando comunicação heterogênea). O processador que minimiza o *EFT* é escolhido para n_i (TOPCUOGLU; HARIRI; WU, 2002). A política de inserção é um diferencial prático do HEFT, reduzindo ociosidade ao encaixar tarefas em *gaps* da agenda.

Vantagens e limitações. O HEFT combina uma visão global (via rank_u) com decisões locais de alocação (via *EFT* e inserção), obtendo bons *makespans* em ampla gama de DAGs heterogêneos. Como limitação, o uso de custos *médios* pode subestimar variações extremas de heterogeneidade e de comunicação.

Complexidade. Seja $v{=}|V|$ o número de tarefas e $p{=}|P|$ o número de processadores. A ordenação por rank_u custa $\mathcal{O}(v \log v)$ após um *DP* linear nas arestas para calcular os ranks. Na fase de alocação, cada tarefa avalia p processadores e verifica posições de inserção em uma agenda de tamanho $\mathcal{O}(v)$, resultando em $\mathcal{O}(v \cdot p \cdot v) = \mathcal{O}(\mathbf{v^2 \cdot p})$ no total (TOPCUOGLU; HARIRI; WU, 2002).

Equações basais.

Seja P o conjunto de processadores. Custos médios:

$$\overline{w}_i = \tfrac{1}{|P|} \sum_{p \in P} w_i^p, \qquad \overline{c}_{i,j} = \tfrac{1}{|P|(|P|-1)} \sum_{p \neq q} c_{i,j}^{p,q}.$$

Rank ascendente (do *exit* para o *entry*):

$$\mathrm{rank}_u(i) = \overline{w}_i + \max_{j \in succ(i)} \Big(\overline{c}_{i,j} + \mathrm{rank}_u(j) \Big), \quad \mathrm{rank}_u(\mathrm{exit}) = \overline{w}_{\mathrm{exit}}.$$

Para alocar i no processador p:

$$\mathrm{EST}(i,p) = \max\Big(\mathrm{avail}(p), \max_{k \in pred(i)} (\mathrm{AFT}(k) + c_{k,i}^{p_k,p}) \Big), \quad \mathrm{EFT}(i,p) = \mathrm{EST}(i,p) + w_i^p,$$

usando política de inserção em lacunas.

O Algoritmo 2 apresenta o pseudocódigo do HEFT, adaptado de Topcuoglu, Hariri e Wu (2002).

Pseudocódigo (HEFT, (TOPCUOGLU; HARIRI; WU, 2002)).

Importante destacar que o HEFT utiliza uma política de inserção de tarefas em lacunas de tempo ocioso: se uma tarefa cabe em um intervalo livre entre duas tarefas já

Algoritmo 2 HEFT

1: Computar $\overline{w}_i$ e $\overline{c}_{i,j}$ para todas as tarefas/arestas
2: Computar $\text{rank}_u(i)$ recursivamente (do fim ao início)
3: Ordenar as tarefas por rank_u (não crescente)
4: **para** tarefa i na ordem **faça**
5: **para** processador $p \in P$ **faça**
6: Calcular $\text{EST}(i, p)$ com inserção; $\text{EFT}(i, p) = \text{EST}(i, p) + w_i^p$
7: Alocar i em $p^* = \arg\min_p \text{EFT}(i, p)$; atualizar $\text{AFT}(i)$ e disponibilidades

escalonadas em um processador (respeitando precedências), ela pode ser inserida ali em vez de ao final da fila (SANDOKJI; EASSA, 2019). Esse mecanismo de inserção melhora o aproveitamento do processador e contribui para reduzir o tempo ocioso. A complexidade de tempo do HEFT é $O(v^2 \cdot p)$(SANDOKJI; EASSA, 2019), bem menor que a do DLS para grafos grandes, o que torna o HEFT mais escalável. Apesar de sua simplicidade, o HEFT costuma produzir cronogramas de boa qualidade – de fato, (TOPCUOGLU; HARIRI; WU, 2002) demonstraram experimentalmente que o HEFT superou diversas heurísticas clássicas (e.g., ETF, Mapping Heuristic, Longest Path) e também obteve resultados superiores ao *Dynamic Level Scheduling* (DLS) em vários cenários. Note que o DLS é anterior ao HEFT, tendo sido introduzido por (SIH; LEE, 1993) e amplamente analisado posteriormente, por exemplo, em (HAGRAS; JANEČEK, 2003).

No contexto de LLMs, podemos interpretar as diferentes partes do treinamento (como computação de distintas camadas ou etapas de pipeline) como tarefas de um DAG. O HEFT, ao levar em conta tanto tempos de processamento quanto de comunicação, pode alocar, por exemplo, camadas maiores para GPUs mais rápidas ou com mais memória (minimizando o EFT), enquanto envia camadas menos críticas a GPUs secundárias. Ele tende a equilibrar bem a utilização ao preencher "bolsas" de tempo livre em GPUs entre sincronizações – análogo a preencher o tempo de uma GPU com outras tarefas enquanto ela espera dados. No entanto, um desafio é que o HEFT assume conhecimento prévio dos tempos de cada tarefa em cada processador. Em um treinamento distribuído real, pode ser complexo estimar com precisão o tempo de, digamos, uma etapa de backpropagation em diferentes GPUs (especialmente considerando efeitos de comunicação entre nós). Trabalhos recentes têm estendido ideias do HEFT para cenários de treino/executação em arquiteturas heterogêneas – por exemplo, (MACK et al., 2021) adaptaram o HEFT para escalonamento dinâmico em tempo de execução em sistemas heterogêneos on-chip, incorporando **eficiência energética** além do tempo(MACK et al., 2021). Isso indica que variantes do HEFT podem inspirar escalonadores energeticamente eficientes para clusters de GPUs variadas.

3.1.3 Lookahead Scheduling (*HEFT-LA*)

O *Lookahead Scheduling* — também referido como *HEFT-LA* — foi proposto como uma extensão do HEFT para mitigar decisões localmente gananciosas (BITTENCOURT; SAKELLARIOU; MADEIRA, 2010; TOPCUOGLU; HARIRI; WU, 2002). Mantém as duas fases do HEFT (priorização por *rank* ascendente e alocação com política de inserção), mas, ao decidir o processador da tarefa corrente, antecipa o impacto sobre seus sucessores.

Ideia central (função de custo com *lookahead*). Para cada tarefa n_i e processador p_j, além do $\mathrm{EFT}(n_i, p_j)$, o HEFT-LA estima o efeito nas tarefas sucessoras:

$$\Phi(n_i, p_j) \;=\; \mathrm{EFT}(n_i, p_j) \;+\; \sum_{n_k \in \mathrm{succ}(n_i)} \min_{p_m \in P} \widehat{\mathrm{EFT}}\big(n_k, p_m \,\big|\, n_i \mapsto p_j\big),$$

onde $\widehat{\mathrm{EFT}}(\cdot)$ é o *EFT* previsto do sucessor n_k ao considerar, hipoteticamente, n_i alocado em p_j (recalculando EST com comunicação/transferência de dados e *insertion policy*). O processador escolhido para n_i é aquele que minimiza $\Phi(n_i, p_j)$. Intuitivamente, a função de custo "olha adiante" um passo no grafo, reduzindo colisões futuras em processadores rápidos e aliviando gargalos (BITTENCOURT; SAKELLARIOU; MADEIRA, 2010).

Fase 1 — Prioridade. As tarefas são ordenadas por rank_u (mesma definição do HEFT), em ordem decrescente (TOPCUOGLU; HARIRI; WU, 2002).

Fase 2 — Alocação com *lookahead*. Na ordem de prioridade, para cada tarefa n_i e cada p_j, calcula-se $\Phi(n_i, p_j)$. O cálculo do $\widehat{\mathrm{EST}}$ de cada sucessor leva em conta: (i) disponibilidade de p_m com **política de inserção** (para encaixar a tarefa em lacunas da agenda) e (ii) tempos de comunicação heterogêneos a partir do processador onde n_i estaria (isto é, dependente da escolha de p_j) (BITTENCOURT; SAKELLARIOU; MADEIRA, 2010). Essa pré-visualização das consequências tende a melhorar o *makespan* em DAGs nos quais múltiplas tarefas críticas competem por poucos recursos rápidos.

Vantagens e limitações. O HEFT-LA preserva a visão global do rank_u e, na alocação, evita escolhas que criariam estrangulamentos adiante. Em contrapartida, o custo computacional cresce rapidamente com v e p, tornando-o menos atrativo para DAGs muito grandes ou ambientes com muitos processadores, a não ser que a qualidade do escalonamento compense o overhead (BITTENCOURT; SAKELLARIOU; MADEIRA, 2010).

Complexidade. Sejam $v{=}|V|$ tarefas e $p{=}|P|$ processadores. O ordenamento por rank_u é análogo ao HEFT. O gargalo é a fase de alocação com *lookahead*, que, no pior caso, avalia combinações tarefa–processador–sucessor–processador-do-sucessor com inserção e comunicação recalculadas; o limite superior analisado em (BITTENCOURT; SAKELLARIOU; MADEIRA, 2010) é $\mathcal{O}(\mathbf{v^4\, p^3})$.

Equações basais.

Para profundidade de antevisão $L \geq 1$ (usualmente L=1), custo de lookahead de profundidade 1 para a tarefa corrente i:

$$\mathrm{LA}(i,p) = \mathrm{EFT}(i,p) + \frac{1}{|succ(i)|} \sum_{j \in succ(i)} \min_{q \in P} \Big(\mathrm{EST}(j,q \,|\, i \to p) + w_j^q \Big),$$

onde $\mathrm{EST}(j,q \,|\, i \to p)$ considera a escolha provisória de i em p. Para $L > 1$, simula-se sucessores até profundidade L.

O Algoritmo 3 apresenta o pseudocódigo do HEFT com lookahead (HEFT-LA), adaptado de Bittencourt, Sakellariou e Madeira (2010).

Pseudocódigo (HEFT-LA, (BITTENCOURT; SAKELLARIOU; MADEIRA, 2010)).

Algoritmo 3 HEFT com Lookahead (profundidade L)

1: Priorizar tarefas como no HEFT (por rank_u)
2: **para** tarefa i na ordem **faça**
3: **para** processador $p \in P$ **faça**
4: Atribuir $i \mapsto p$ provisoriamente e calcular $\mathrm{EFT}(i,p)$
5: Simular (até profundidade L) a alocação dos sucessores com HEFT, obtendo custo agregado C_L
6: Definir $\mathrm{LA}(i,p) = \mathrm{EFT}(i,p) + C_L$
7: Fixar $p^* = \arg\min_p \mathrm{LA}(i,p)$ e efetivar a alocação; atualizar estado

O preço a pagar por essa melhoria de qualidade de escalonamento é um **custo computacional significativamente maior**. O algoritmo Lookahead realiza análises combinatórias mais complexas, o que resulta em complexidade de tempo bem superior à do HEFT. Estudos indicam que sua complexidade é da ordem de $O(v^3 \cdot p^2)$ em certos cenários, aproximando-se de **complexidade quadrática de ordem quatro (quartic)** no número de tarefas no pior caso(NIYOM; SOPHATSATHIT; LURSINSAP, 2016) (ARABNEJAD; BARBOSA, 2013). Em outras palavras, embora não haja um consenso simples, frequentemente cita-se que o Lookahead possui complexidade polinomial de grau 4 em v. Isso o torna menos viável para grafos muito grandes (centenas ou milhares de tarefas) sem alguma otimização adicional.

Do ponto de vista de desempenho, o Lookahead de fato consegue melhorar o escalonamento em relação ao HEFT em muitos casos, especialmente em cenários onde as heterogeneidades de processamento e comunicação são acentuadas. Por exemplo, em experimentos com grafos de até 100 tarefas, o algoritmo Lookahead obteve makespans menores que o HEFT tradicional e que outros concorrentes, aproximando-se bastante do ótimo em diversos casos(ARABNEJAD; BARBOSA, 2013). Além disso, um estudo reporta que o Lookahead tende a produzir um melhor balanceamento de carga entre os

processadores do que o PEFT (descrito adiante), justamente por distribuir algumas tarefas de forma mais antecipativa mesmo que isso não reduza imediatamente o tempo daquela tarefa(SANDOKJI; EASSA, 2019). Esse melhor balanceamento reflete em maior **uso de recursos** global. Contudo, conforme o tamanho do DAG cresce, os ganhos de Lookahead sobre algoritmos quadráticos (como HEFT e PEFT) tendem a diminuir, ao passo que o custo de execução do escalonador aumenta. (ARABNEJAD; BARBOSA, 2013) observaram que para grafos maiores (centenas de nós), o algoritmo PEFT conseguia igualar ou superar o Lookahead em qualidade de escalonamento, apesar de sua complexidade menor, tornando o Lookahead menos atraente nesses cenários maiores(ARABNEJAD; BARBOSA, 2013)

Em aplicações de treinamento de LLMs, o Lookahead seria conceitualmente útil se quisermos explorar ao máximo a flexibilidade de alocar partes do modelo em diferentes GPUs heterogêneas. Por exemplo, imagine decidir em qual GPU colocar um determinado conjunto de camadas: uma GPU mais antiga pode terminar essa parte ligeiramente mais devagar, mas talvez facilite que as próximas camadas (sucessoras) sejam já computadas em outra GPU sem esperar pela liberação de uma GPU mais potente. O Lookahead poderia identificar essa situação e decidir por essa alocação "antecipativa" visando o melhor desempenho global, em vez de insistir que todas as partes rodem na GPU mais rápida (o que poderia provocar filas de espera nela). Entretanto, implementar esse algoritmo no contexto de treinamento distribuído requer modelar cuidadosamente o DAG de treinamento (incluindo comunicações de gradientes, sincronizações de parâmetros, etc.) e pode incorrer em alto overhead de planejamento. Na prática, devido ao seu custo, o Lookahead puro não é muito mencionado em sistemas de ML; em vez disso, ideias de lookahead são incorporadas de forma heurística em frameworks (por exemplo, algumas abordagens de paralelismo pipeline utilizam heurísticas para balancear etapas considerando passos futuros).

3.1.4 Predict Earliest Finish Time (PEFT)

O *Predict Earliest Finish Time* (PEFT) (ARABNEJAD; BARBOSA, 2013) é uma heurística de *list scheduling* que aprimora o HEFT mantendo a **mesma ordem de complexidade**. A ideia-chave é a **Tabela de Custos Otimistas** (*Optimistic Cost Table*, OCT), que antecipa — de forma otimista — o menor tempo possível para concluir o *subgrafo* a partir de cada tarefa em cada processador.

OCT (definição e preenchimento). Para cada tarefa n_i e processador p_j, a OCT é calculada em ordem topológica reversa como:

$$\mathrm{OCT}(n_i, p_j) \;=\; w_{i,j} \;+\; \max_{n_k \in \mathrm{succ}(n_i)} \Big(\min_{p_m \in P} \big(\widehat{c}_{i,k}(j,m) + \mathrm{OCT}(n_k, p_m) \big) \Big),$$

onde $w_{i,j}$ é o tempo de computação de n_i em p_j e $\widehat{c}_{i,k}(j,m)$ estima o custo de comunicação

na aresta (i, k) ao enviar de p_j para p_m. Para tarefas sem sucessores, $\mathrm{OCT}(n_i, p_j) = w_{i,j}$. A OCT fornece, assim, um limite inferior "otimista" do tempo de conclusão do caminho restante a partir de n_i.

Prioridade ($\mathbf{rank_{OCT}}$). O PEFT define um *rank* baseado na OCT, por exemplo pela média sobre processadores:

$$\mathrm{rank}_{\mathrm{OCT}}(n_i) \;=\; \frac{1}{p} \sum_{p_j \in P} \mathrm{OCT}(n_i, p_j),$$

e ordena as tarefas em ordem decrescente desse valor. Essa prioridade captura melhor a "criticidade global" do nó do que o uso de custos médios do HEFT, mitigando decisões míopes.

Alocação (seleção de processador). Seguindo a ordem por $\mathrm{rank}_{\mathrm{OCT}}$, para cada tarefa n_i avaliam-se todos os processadores p_j e computa-se o $\mathrm{EFT}(n_i, p_j) = \mathrm{EST}(n_i, p_j) + w_{i,j}$ com **política de inserção** (preenchendo lacunas na agenda) e comunicação heterogênea; escolhe-se o p_j que minimiza o EFT. A diferença prática em relação ao HEFT está na *priorização*: a OCT guia a ordem de escalonamento para beneficiar o *makespan* global, podendo, deliberadamente, evitar alocar uma tarefa no processador de término "localmente" mais rápido quando isso prejudicaria sucessores críticos (ARABNEJAD; BARBOSA, 2013).

Complexidade. Sejam $v{=}|V|$ tarefas e $p{=}|P|$ processadores. O preenchimento da OCT em ordem topológica reversa custa $\mathcal{O}(e \cdot p)$ (majorado por $\mathcal{O}(v^2 \cdot p)$), a ordenação custa $\mathcal{O}(v \log v)$, e a fase de alocação com inserção, ao avaliar p processadores e potenciais posições na agenda ao longo do escalonamento, resulta em $\mathcal{O}(v^2 \cdot p)$. Portanto, o custo total do PEFT é $\mathcal{O}(\mathbf{v^2 \cdot p})$, *igual* ao do HEFT (ARABNEJAD; BARBOSA, 2013; TOPCUOGLU; HARIRI; WU, 2002).

Equações basais.

Tabela de Custos Otimistas (OCT), de trás para frente:

$$\mathrm{OCT}(i, p) = w_i^p + \max_{j \in succ(i)} \min_{q \in P} \Big(c_{i,j}^{p,q} + \mathrm{OCT}(j, q) \Big), \qquad \mathrm{OCT}(\mathrm{exit}, p) = 0.$$

Ordenação por:

$$\mathrm{rank}_{oct}(i) = \max_{p \in P} \mathrm{OCT}(i, p).$$

A seleção do processador segue o menor $\mathrm{EFT}(i, p)$ (com inserção), como no HEFT.

O Algoritmo 4 apresenta o pseudocódigo do PEFT, adaptado de Arabnejad e Barbosa (2013).

Pseudocódigo (PEFT, (ARABNEJAD; BARBOSA, 2013)).

Algoritmo 4 PEFT

1: Pré-computar $\mathrm{OCT}(i, p)$ para todos i, p (do fim ao início)
2: Calcular $\mathrm{rank}_{oct}(i)$; ordenar tarefas por ordem não crescente
3: **para** tarefa i na ordem **faça**
4: **para** processador $p \in P$ **faça**
5: Calcular $\mathrm{EST}(i, p)$ e $\mathrm{EFT}(i, p)$ (política de inserção)
6: Alocar i em $p^* = \arg\min_p \mathrm{EFT}(i, p)$; atualizar $\mathrm{AFT}(i)$ e disponibilidades

Esse comportamento "não-guloso" distingue o PEFT. Uma vantagem importante do PEFT é que sua **complexidade permanece** $O(v^2 \cdot p)$**,** igual à do HEFT(ARABNEJAD; BARBOSA, 2013). O cálculo da OCT adiciona um overhead linear nas arestas (proporcional a $p(e + v)$, com e arestas)(ARABNEJAD; BARBOSA, 2013), mas isso não altera a ordem assintótica dominante. Portanto, o PEFT oferece uma melhora de desempenho de escalonamento sem sacrificar escalabilidade computacional. Nos resultados reportados, o PEFT efetivamente superou tanto o HEFT quanto o Lookahead em muitos cenários, produzindo escalonamentos com **menor makespan** do que o HEFT na maioria dos casos. Para grafos pequenos (até 100 tarefas), o PEFT tem desempenho semelhante ao do Lookahead, e para grafos maiores ele se saiu melhor, justamente porque o Lookahead não consegue escalar tão bem. Em aproximadamente 72% das execuções testadas, o PEFT obteve o melhor escalonamento dentre os algoritmos comparados. Além disso, o PEFT manteve bom **balanceamento de carga** e **uso de recursos**, aproximando-se da qualidade de algoritmos mais complexos (porém de complexidade quadrática apenas)(ARABNEJAD; BARBOSA, 2013).

No contexto de treinamento distribuído de LLMs, o PEFT seria uma escolha promissora para escalonar tarefas se pudermos modelar o treinamento como um DAG estático antes da execução. Por exemplo, considere um pipeline de treinamento com várias etapas (pré-processamento de dados, cálculo do grafo forward para diferentes partes do modelo, sincronização de gradientes, atualização de parâmetros, etc.) distribuídas entre GPUs com diferentes capacidades. O PEFT, alimentado com uma tabela de custos otimistas (que poderiam ser obtidos, por exemplo, medindo latências de partes do modelo em cada tipo de GPU), poderia gerar um plano de execução que minimiza o tempo total por passo de treino. Uma característica útil é que o PEFT provavelmente evitaria congestionar sempre a GPU mais poderosa com todas as tarefas: se a tabela indicar que é melhor colocar algumas camadas numa GPU secundária para que a GPU principal possa em paralelo processar outra parte crítica, o algoritmo tomará essa decisão. Assim, espera-se um bom balanceamento entre utilizar GPUs topo de linha para as partes realmente mais pesadas, e aproveitar GPUs mais lentas para trabalho em paralelo nas partes menos críticas. Em termos práticos, há desafios – por exemplo, a predição otimista pode não refletir perfeita-

mente o comportamento real em tempo de execução, especialmente sob contensão de rede ou variações – mas técnicas de PEFT poderiam ser incorporadas em orchestradores de treinamento. De fato, a literatura recente em sistemas de ML distribuído já esboça aproximações disso: por exemplo, o sistema **FlashFlex** (YAN et al., 2024) formaliza a alocação de partes do treinamento de LLMs heterogeneamente como um problema de otimização em grafo e utiliza um particionamento hierárquico do grafo do modelo para distribuir as computações de forma quase ótima(YAN et al., 2024). Embora não seja exatamente o PEFT, o espírito é semelhante – usar uma visão global (neste caso, um particionamento do grafo de compute do LLM) para alocar cargas em diferentes GPUs de modo a **maximizar o uso de recursos e reduzir o makespan**. Esses resultados ilustram o potencial de técnicas de escalonamento inteligente nesse domínio.

3.1.5 Improved HEFT (IHEFT)

O *Improved HEFT* (IHEFT) (ALEBRAHIM; AHMAD, 2016) preserva o esqueleto do HEFT — duas fases de **priorização** por *rank* e **alocação** por *EFT* com política de inserção — porém torna a heurística mais *comunicação-ciente* e *criticidade-ciente*. Em síntese, (i) ajusta o cálculo de prioridade incorporando custos de comunicação (visão local e global) para evitar enviesar sempre tarefas "grandes", e (ii) modifica a seleção de processador com um mecanismo de *limiar aleatorizado* para decidir quando "cruzar" entre processadores (p.ex., CPUGPU), reduzindo *crossovers* desnecessários e gargalos futuros. :contentReference[oaicite:0]index=0

Fase 1 — Prioridade aprimorada. O peso usado no *rank* é revisado para considerar, além do custo de execução, os custos de comunicação ao longo do caminho restante (componentes local e global), priorizando nós críticos sem supervalorizar apenas tarefas com maior $w_{i,j}$. Isso melhora a ordenação inicial frente a variações de heterogeneidade e CCR. :contentReference[oaicite:1]index=1

Fase 2 — Alocação com decisão de crossover. Mantendo a política de inserção, para cada tarefa n_i e candidato p_j calcula-se o *EFT*; a decisão de migrar dados/execução entre tipos de processador usa um *threshold* aleatorizado que explicitamente incorpora custos de comunicação, explorando melhor o espaço de soluções e evitando migrações de alto custo. :contentReference[oaicite:2]index=2

Complexidade. Seja $v{=}|V|$ o número de tarefas e $p{=}|P|$ o número de processadores. O IHEFT mantém a mesma ordem assintótica de HEFT e PEFT, com custo total $\mathcal{O}(\mathbf{v^2} \cdot \mathbf{p})$ (TOPCUOGLU; HARIRI; WU, 2002; ARABNEJAD; BARBOSA, 2013). Em particular, ALEBRAHIM; AHMAD reportam explicitamente $O(t^2p)$ para a heurística proposta (com $t{=}v$), isto é, mesma classe de complexidade dos baselines.

Equações basais.

Classificação de criticidade. A partir de custos médios, computam-se tempos de início/término mais cedo e mais tarde (AEST/ALST); um nó i é crítico se $\mathrm{ALST}(i) = \mathrm{AEST}(i)$.

$$\mathrm{AEST}(i) = \overline{w}_i + \max_{k \in pred(i)} \Big(\mathrm{AEST}(k) + \overline{c}_{k,i}\Big), \qquad \mathrm{ALST}(i) = \min_{j \in succ(i)} \Big(\mathrm{ALST}(j) - \overline{c}_{i,j}\Big) - \overline{w}_i.$$

Afinidade de comunicação. Para tarefa i e processador p:

$$\mathrm{Aff}(i,p) = \sum_{k \in pred(i)} \mathbf{1}[\sigma(k) = p] \cdot d_{k,i},$$

onde $d_{k,i}$ é o volume de dados de (k,i). Maior Aff sugere menor custo de comunicação esperado.

O Algoritmo 5 apresenta o pseudocódigo do IHEFT, adaptado de AlEbrahim e Ahmad (2016). **Pseudocódigo (IHEFT, (ALEBRAHIM; AHMAD, 2016)).**

Algoritmo 5 IHEFT

1: Calcular $\overline{w}, \overline{c}$; rank_u como no HEFT
2: Identificar nós críticos por AEST/ALST
3: Ordenar tarefas por rank_u (não crescente); desempatar favorecendo (crítico $\succ$ não crítico) e maior grau de saída
4: **para** tarefa i na ordem **faça**
5: **para** processador $p \in P$ **faça**
6: Calcular $\mathrm{EST}(i,p)$ e $\mathrm{EFT}(i,p)$ com inserção
7: Selecionar p^* por ordem lexicográfica:

$$\min \Big(\mathrm{EFT}(i,p),\ -\mathrm{Aff}(i,p),\ \text{maior tempo livre em } p \Big)$$

8: Alocar $i \mapsto p^*$; atualizar AFT, σ, disponibilidades

Complexidade. Overhead adicional para AEST/ALST e afinidade, mas mesma ordem assintótica do HEFT ($O(v^2 p)$) na prática.

No treinamento distribuído de LLMs, o DAG típico do *step* contém subgrafos de *forward*/*backward* por bloco (atenção, MLP, normalizações), seguidos de sincronizações (p. ex., *all-reduce* de gradientes) e atualização do otimizador. O IHEFT é especialmente adequado quando há **heterogeneidade de hardware e de interconexão** (GPUs com gerações distintas, enlaces NVLink/PCIe/Ethernet mistos): (i) a **detecção de nós críticos** concentra recursos mais rápidos (e janelas de execução) nas cadeias que determinam o tempo do *step* – tipicamente as últimas retropropagações e sincronizações – reduzindo bolhas; (ii) a **afinidade de comunicação** tende a manter sucessões com alto tráfego (ativação/gradiente) no mesmo dispositivo ou *island* de alta banda, reduzindo latências e *stall* por espera de dados. Na prática, estima-se Aff a partir de perfis de bytes/latência

entre estágios (por exemplo, tamanhos de tensores de atenção e gradientes), e proíbe-se alocações que violem memória. O ganho é maior em *pipeline/tensor parallel* com shards assimétricos, pois o IHEFT evita *ping-pong* entre GPUs rápidas e lentas, equilibrando uso sem penalizar o caminho crítico. Limitações: exige estimativas razoáveis de w_i^p e $c_{i,j}^{p,q}$; variação de carga dinâmica pode requerer realimentação periódica ou reescalonamento em janelas.

3.1.6 Improved PEFT (IPEFT)

O *Improved PEFT* (IPEFT) (ZHOU et al., 2016) estende o PEFT substituindo a única **OCT** por duas tabelas: uma **PCT** (pessimista) e uma **CNCT** (sensível a nós críticos). A primeira é usada para **priorizar** tarefas; a segunda, para **selecionar** o processador minimizando um *EFT* penalizado por comunicação rumo a sucessores críticos — com exceção para *pais de nós críticos* (*CNP*), nos quais a penalidade é desativada para não retardar o caminho crítico.

PCT — Pessimistic Cost Table (prioridade). Em ordem topológica reversa, para cada tarefa n_i e processador p_j,

$$\mathrm{PCT}(n_i, p_j) = w_{i,j} + \max_{n_k \in \mathrm{succ}(n_i)} \Big(\max_{p_m \in P} \big(\widehat{c}_{i,k}(j,m) + \mathrm{PCT}(n_k, p_m) \big) \Big),$$

onde $w_{i,j}$ é o tempo de computação em p_j e $\widehat{c}_{i,k}(j,m)$ é o custo de comunicação de $p_j \to p_m$ na aresta (i,k). Para folhas, $\mathrm{PCT}(n_i,p_j){=}w_{i,j}$. Define-se então o $\mathrm{rank}_{\mathrm{PCT}}(n_i)$ (p.ex., média de PCT sobre p) e ordenam-se as tarefas em ordem decrescente.

CNCT — Critical-Node-aware Cost Table (seleção). Seja $\mathcal{C}$ o conjunto de *nós críticos* (obtido por uma pontuação de criticidade baseada no impacto no caminho até o *exit*). Para cada n_i, p_j,

$$\mathrm{CNCT}(n_i, p_j) = \sum_{n_k \in \mathrm{succ}(n_i)} \min_{p_m \in P} \Big(\gamma_k \, \widehat{c}_{i,k}(j,m) \Big), \quad \gamma_k = \begin{cases} > 1, & n_k \in \mathcal{C} \\ 0 \text{ ou } 1, & \text{caso contrário} \end{cases}$$

penalizando comunicações que "empurram" dados para sucessores críticos. Na alocação, para cada tarefa n_i (na ordem por $\mathrm{rank}_{\mathrm{PCT}}$) e processador candidato p_j, calcula-se

$$\Psi(n_i, p_j) \;=\; \mathrm{EFT}(n_i, p_j) \;+\; \lambda \cdot \mathrm{CNCT}(n_i, p_j),$$

com $\lambda{>}0$ controlando a influência da penalidade; escolhe-se o p_j que minimiza Ψ. Se n_i é *CNP* (pai de nó crítico), usa-se $\lambda{=}0$ (sem penalidade), a fim de não atrasar o avanço do caminho crítico. Mantém-se a **política de inserção** para preencher lacunas na agenda.

Vantagens e limitações. Comparado ao PEFT, o IPEFT tende a reduzir *ping-pong* de dados em torno de sucessores críticos e a suavizar gargalos, com ganhos de *makespan* mais visíveis em CCRs altos ou DAGs com forte assimetria de comunicação. O

custo adicional vem do cálculo de duas tabelas e da penalização na seleção, porém sem alterar a ordem assintótica.

Complexidade. Sejam $v{=}|V|$ tarefas e $p{=}|P|$ processadores. O preenchimento da PCT e da CNCT é $\mathcal{O}(e \cdot p)$ cada (majorado por $\mathcal{O}(v^2 \cdot p)$); a ordenação custa $\mathcal{O}(v \log v)$; e a alocação com inserção, ao avaliar p processadores ao longo do escalonamento, resulta em $\mathcal{O}(v^2 \cdot p)$. Logo, o IPEFT mantém a mesma classe de complexidade de HEFT/PEFT, com $\mathcal{O}(\mathbf{v^2 \cdot p})$ (ZHOU et al., 2016; ARABNEJAD; BARBOSA, 2013; TOPCUOGLU; HARIRI; WU, 2002).

Equações basais.

Tabela Pessimista (PCT):

$$\mathrm{PCT}(i,p) = w_i^p + \max_{j \in succ(i)} \min_{q \in P} \Big(c_{i,j}^{p,q} + \mathrm{PCT}(j,q) \Big), \qquad \mathrm{PCT}(\mathrm{exit},p) = 0.$$

Nós críticos e pais críticos. Via AEST/ALST, defina $CN = \{i : \mathrm{ALST}(i) = \mathrm{AEST}(i)\}$ e $CNP = \{i : \exists j \in CN, (i,j) \in E\}$. **Tabela CNCT (penalidade por sucessores críticos):**

$$\mathrm{CNCT}(i,p) = \max_{j \in succ(i) \cap CN} \min_{q \in P} \Big(c_{i,j}^{p,q} + \mathrm{PCT}(j,q) \Big),$$

com $\mathrm{CNCT}(i,p) = 0$ se $succ(i) \cap CN = \emptyset$. **Prioridade e seleção.** Prioridade:

$$\mathrm{rank}_{\mathrm{IPEFT}}(i) = \frac{1}{|P|} \sum_{p \in P} \mathrm{PCT}(i,p) + \overline{w}_i.$$

Seleção em p:

$$\mathrm{EFT}_{\mathrm{CNCT}}(i,p) = \begin{cases} \mathrm{EFT}(i,p), & \text{se } i \in CNP, \\ \mathrm{EFT}(i,p) + \mathrm{CNCT}(i,p), & \text{caso contrário.} \end{cases}$$

O Algoritmo 6 apresenta o pseudocódigo do IPEFT, adaptado de Zhou et al. (2016). **Pseudocódigo (IPEFT, (ZHOU et al., 2016)).**

Algoritmo 6 IPEFT

1: Pré-computar $\mathrm{PCT}(i,p)$ (do fim ao início)
2: Calcular AEST/ALST; obter CN e CNP; computar $\mathrm{CNCT}(i,p)$
3: Calcular $\mathrm{rank}_{\mathrm{IPEFT}}(i)$; ordenar tarefas (não crescente)
4: **para** tarefa i na ordem **faça**
5: **para** processador $p \in P$ **faça**
6: Calcular $\mathrm{EST}(i,p)$ e $\mathrm{EFT}(i,p)$ com inserção
7: Definir $\mathrm{EFT}_{\mathrm{CNCT}}(i,p)$ conforme acima
8: Alocar i em $p^* = \arg\min_p \mathrm{EFT}_{\mathrm{CNCT}}(i,p)$; atualizar estado

Complexidade. Mantém a escalabilidade de HEFT/PEFT ($O(v^2p)$) com pré-cálculos adicionais moderados.

Em LLMs distribuídos, o gargalo do *step* costuma residir no **caminho crítico** que conecta as últimas retropropagações, as sincronizações de gradiente (*all-reduce*/*reduce-scatter*) e a atualização do otimizador. O IPEFT modela isso de forma explícita: a PCT prioriza tarefas cujo atraso propagaria até o *exit*, enquanto a CNCT **penaliza escolhas de processador que retardem sucessores críticos**, preservando exceções para *pais de nós críticos* (que devem ser acelerados). Em termos práticos, isso resulta em (i) **reservar GPUs mais rápidas** para blocos críticos (p. ex., camadas finais, sincronizações e atualização), (ii) **alocar blocos não críticos** (embeddings, parte de MLPs menos custosas, pré/pós-processamento) em GPUs mais lentas, e (iii) **reduzir bolhas** no estágio de comunicação por antecipar a conclusão dos pais de nós críticos. O mecanismo é compatível com *data*/*tensor*/*pipeline parallel*: a CNCT naturalmente "puxa" shards que alimentam as sincronizações para recursos de maior desempenho ou melhor conectividade (NVLink/Infiniband), enquanto a PCT evita miopia local ao estimar o custo *até* o término do passo. Como limitações, o modelo permanece estático e sensível à qualidade das estimativas de w_i^p e $c_{i,j}^{p,q}$; para cargas variáveis (flutuações de *dataloader*, *stragglers* de rede), recomenda-se recalcular as tabelas periodicamente ou operar em janelas deslizantes com reescalonamento leve.

3.2 Geração Sintética de Grafos de Tarefas

Para avaliar as heurísticas em topologias alinhadas à morfologia discutida pelo *Predict Earliest Finish Time* (PEFT) (ARABNEJAD; BARBOSA, 2013), geramos instâncias sintéticas que chamamos de **PEFT-like**. O objetivo não é reproduzir *ipsis litteris* os grafos do artigo, mas *induzir* propriedades estruturais e de custo semelhantes (estratificação em camadas, abertura moderada nas camadas iniciais, convergência para *sinks* e custos de comunicação crescentes com a profundidade), preservando **reprodutibilidade** via uma única semente pseudoaleatória.

3.2.1 Forma do DAG

Sejam N o número de tarefas e L o número de camadas (por padrão, $L \approx 6$). As tarefas são distribuídas quasi-uniformemente entre as camadas, assegurando uma ordenação topológica natural. As arestas conectam *apenas* para a frente:

- Cada tarefa u na camada $\ell \in \{0, \ldots, L-2\}$ recebe grau de saída $d_{\text{out}}(u) \sim \text{Unif}\{1, 2, 3\}$.
- Sucessores são escolhidos na próxima camada $\ell{+}1$; com pequena probabilidade, permite-se um salto para $\ell{+}2$ (se $\ell{+}2 < L$), evitando ciclos.
- A topologia abre moderadamente nas primeiras camadas e converge para poucos *sinks* nas camadas finais, garantindo conectividade de *entry* a *exit*.

- Toda aleatoriedade é condicionada a uma única `seed` global, garantindo reprodutibilidade.

3.2.2 Pesos de comunicação (arestas)

Para cada aresta (u, v) sorteia-se um *payload* base $d_{u,v}$:

$$d_{u,v} \sim \begin{cases} 0, & \text{com prob. } 0{,}15, \\ \text{UnifInt}[5, 60], & \text{caso contrário,} \end{cases}$$

e aplica-se um **fator de profundidade** crescente com a camada de origem $\ell(u)$:

$$f(\ell) \;=\; 1 \;+\; 0{,}6 \cdot \frac{\ell}{L-1}, \qquad \text{peso}_{\text{comm}}(u, v) \;=\; \text{round}\Big(d_{u,v} \cdot f(\ell(u))\Big).$$

3.2.3 Tempos de execução (heterogeneidade com afinidade)

Cada tarefa i recebe um tempo base

$$u_i \sim \text{Unif}[8, 40].$$

Cada processador p recebe um multiplicador de desempenho

$$\gamma_p \sim \text{Unif}[0{,}75,\ 1{,}25] \quad \text{(menor = mais rápido)},$$

e definimos uma **afinidade tarefa–processador** determinística e reprodutível:

$$\delta_{i,p} = \begin{cases} 0{,}7, & \text{se } (i \bmod 17) = (p \bmod 17), \\ 1, & \text{caso contrário.} \end{cases}$$

O tempo de execução final por tarefa e processador é

$$w_i^p = \max\Big(1,\ \text{round}(u_i \cdot \gamma_p \cdot \delta_{i,p})\Big).$$

3.2.4 Potência/Energia (métrica derivada, não-decisória)

A potência por processador é modelada como fracamente correlacionada ao inverso do multiplicador (dispositivos mais "rápidos" tendem a consumir um pouco mais), com ruído pequeno:

$$P_p \;=\; \kappa \cdot \gamma_p^{-0{,}8} \cdot (1 + \epsilon_p), \qquad \epsilon_p \sim \mathcal{N}(0, \sigma^2),\ \sigma \approx 0{,}03.$$

Para capturar o efeito da **afinidade** muito benéfica, se

$$w_i^p \;<\; 0{,}85 \cdot \frac{1}{|\mathcal{P}|} \sum_{q \in \mathcal{P}} w_i^q,$$

aplica-se um *bônus* de potência de $\approx 5\%$: $P_p \leftarrow 1{,}05 \cdot P_p$ para a execução daquela tarefa. A **energia** é apurada *após* o escalonamento (não influencia as decisões):

$$E_{\text{total}} = \sum_{i \in V} P_{\sigma(i)} \cdot w_i^{\sigma(i)}.$$

3.2.5 Matriz de largura de banda (recursos)

A matriz $B \in \mathbb{R}^{P \times P}$ é simétrica, com diagonal nula (sem custo intra-dispositivo) e heterogeneidade simples:

$$B_{a,b} = \begin{cases} 0, & a = b, \\ 2, & a \neq b \text{ com prob. } 0{,}10, \\ 1, & \text{caso contrário}, \end{cases} \qquad B_{a,b} = B_{b,a}.$$

Os custos efetivos de comunicação usados pelas heurísticas seguem

$$c_{m,i}^{a,b} = \lambda_{a,b} + \frac{d_{m,i}}{B_{a,b}},$$

com $\lambda_{a,b}$ (latência) opcionalmente fixada a um valor médio.

3.2.6 Parâmetros padrão

Tabela 1 – Parâmetros padrão das instâncias PEFT-like.

Componente	**Valor/Distribuição**
Camadas L	≈ 6 (configurável)
Distribuição por camada	quase uniforme (ajuste do resto nas camadas iniciais)
Grau de saída	Unif$\{1,2,3\}$
Salto de camada	alvo $\ell{+}1$; ocasionalmente $\ell{+}2$
$P(d{=}0)$	0,15
Peso de aresta d	UnifInt$[5,60]$ (se $d \neq 0$)
Fator de profundidade	$f(\ell) = 1 + 0{,}6 \cdot \frac{\ell}{L-1}$
Base de execução u_i	Unif$[8,40]$
Multiplicador γ_p	Unif$[0{,}75, 1{,}25]$
Afinidade $\delta_{i,p}$	0,7 se $(i \bmod 17) = (p \bmod 17)$; senão 1
Rounding e piso	$w_i^p = \max(1, \mathrm{round}(\cdot))$
Potência P_p	$\kappa \cdot \gamma_p^{-0{,}8} \cdot (1+\epsilon_p)$, $\epsilon_p \sim \mathcal{N}(0, 0{,}03^2)$
Bônus de potência	$+5\%$ se $w_i^p < 0{,}85 \cdot \text{média}_q\, w_i^q$
Banda $B_{a,b}$	1 (90%) ou 2 (10%), simétrica; $B_{a,a} = 0$
Seed	única por execução (fixa)

3.2.7 Pseudocódigo (alto nível)

Algoritmo 7 Gerar instâncias `PEFT-like`$(N, P, L, \texttt{seed})$

Fixar gerador pseudoaleatório com `seed`
Particionar N tarefas em L camadas quasi-uniformes; criar *entry*/*exit*
para cada tarefa u nas camadas 0 a $L{-}2$ **faça**
    Sortear $d_{\text{out}}(u) \in \{1,2,3\}$; escolher sucessores em $\ell{+}1$ (salto para $\ell{+}2$ ocasional)
    para cada aresta (u,v) **faça**
        Sortear $d_{u,v} \in \{0\ (15\%),\ \text{UnifInt}[5,60]\}$ e aplicar $f(\ell(u))$
Para cada tarefa i: $u_i \sim \text{Unif}[8,40]$
Para cada processador p: $\gamma_p \sim \text{Unif}[0{,}75, 1{,}25]$ e $P_p \leftarrow \kappa \cdot \gamma_p^{-0{,}8} \cdot (1+\epsilon_p)$
para cada par (i,p) **faça**
    $\delta_{i,p} \leftarrow 0{,}7$ se $(i \bmod 17) = (p \bmod 17)$, senão 1
    $w_i^p \leftarrow \max(1, \mathrm{round}(u_i \cdot \gamma_p \cdot \delta_{i,p}))$
    se $w_i^p < 0{,}85 \cdot \frac{1}{P} \sum_q w_i^q$ **então**
        $P_p \leftarrow 1{,}05 \cdot P_p$
Construir B: $B_{a,a} = 0$; para $a \neq b$, $B_{a,b} = B_{b,a} \in \{2\ (10\%), 1\ (90\%)\}$
Salvar CSVs: conectividade $(d_{u,v})$, execução (w_i^p), banda (B), potência (P_p opcional)

3.2.8 Reprodutibilidade e sanidade

- **Reprodutibilidade**: uma única `seed` governa topologia, pesos de aresta, tempos de execução, banda e potência.

- **Sanidade**: rejeitar instâncias com nós isolados, desconexas (*entry*$\not\to$*exit*) ou com ciclos (checado por ordenação topológica).
- **Domínios**: garantir $w_i^p \geq 1$, $B_{a,b} \in \{0, 1, 2\}$ e pesos de aresta não negativos.

3.2.9 Ajuste opcional de CCR

Quando desejado, ajustamos a razão *compute/communication* (CCR) alvo escalando os pesos de aresta:

$$\text{CCR} \approx \frac{\overline{C}}{\overline{M}}, \quad \overline{C} = \frac{1}{N} \sum_i \min_p w_i^p, \quad \overline{M} = \frac{1}{|E|} \sum_{(u,v)\in E} \frac{d_{u,v}}{\overline{B}},$$

e aplicamos $d_{u,v} \leftarrow \alpha \cdot d_{u,v}$ com α escolhido para aproximar a CCR desejada (mantendo limites mínimos/máximos).

3.2.10 Integração com os algoritmos

Os arquivos gerados são consumidos diretamente pelas implementações via linha de comando:

```
python compare_same_dataset.py \
  --dag  graphs/{tag}_task_connectivity.csv \
  --exec graphs/{tag}_task_exe_time.csv \
  --bw   graphs/{tag}_resource_BW.csv \
  --algos HEFT,PEFT,DLS,HEFT-LA,IHEFT,IPEFT --report
```

3.2.10.0.1 Disponibilização e licença (MIT).

Todo o código-fonte deste projeto – geradores de instâncias, implementações das heurísticas, scripts de experimentos e de plotagem – está disponível em repositório *open-source* sob licença MIT. A **descrição detalhada do projeto e da implementação**, bem como o **passo a passo para executar os testes**, encontram-se no **Apêndice A** ("Reprodutibilidade e Repositório").

3.2.10.0.2 Observação.

A energia é usada **apenas** como métrica pós-execução e *não* altera as decisões das heurísticas.

3.2.10.1 Síntese

Em síntese temos o seguinte funcionamento:

Reprodutibilidade. Todos os componentes são gerados com a mesma semente pseudoaleatória e deslocamentos fixos: semente para o DAG; semente+1 para tempos de execução; semente+2 para potência; semente+3 para banda. Isso garante resultados determinísticos.

Execução. Tempos base aleatórios por tarefa multiplicados por um fator de desempenho específico do processador; uma regra de afinidade (congruência módulo 17) reduz o tempo quando há correspondência.

Potência. Derivada aproximadamente do inverso do desempenho com pequena variação; há um ligeiro aumento quando a tarefa se torna mais rápida.

Banda. Matriz simétrica (diagonal nula) de larguras de banda entre recursos, com valores 1 e, ocasionalmente, 2; gerada com o deslocamento de semente correspondente.

Conectividade (DAG). Matriz de adjacência ponderada dos arcos, definindo as dependências e pesos de comunicação entre tarefas.

3.3 Herança nas tabelas de custos

A cada aumento de número de processadores, copiamos exatamente os dados já existentes dos processadores anteriores e só "criamos" valores novos para os processadores adicionais. As tarefas e ligações entre elas não mudam; apenas estendemos a tabela, mantendo intacta a parte antiga. A matriz de comunicação reaproveita o bloco antigo no canto superior esquerdo e gera linhas/colunas só para os novos processadores. Assim, comparar tempos entre 8, 16 e 32 processadores é justo porque nada do que já existia foi alterado.

4 Experimentos

Este capítulo apresenta a avaliação empírica das heurísticas **DLS**, **HEFT**, **HEFT-LA**, **PEFT**, **IHEFT** e **IPEFT** sobre DAGs gerados pelo procedimento definido em 2.5.

O foco é quantificar, em diferentes escalas de problema — **tarefas (N)** — e capacidades de hardware definidas pelo número de **processadores (P)**, o impacto de cada heurística nas métricas: **makespan** (MS), **load balance**, **tempo médio de espera (waiting time)** ($\overline{\mathrm{WT}}$) e **energia derivada** (E_{total}).

A experimentação foi estruturada em **cinco cenários de tamanho**, cada qual avaliado em três configurações de paralelismo ($P \in \{8, 16, 32\}$). A Tabela 2 sintetiza os cinco cenários avaliados neste capítulo, variando o número de tarefas (N) e o número de processadores (P).

Tabela 2 – Cenários de experimentos

Cenário	**Configurações**
Cenário 1	N=256 tarefas; $P \in \{8, 16, 32\}$
Cenário 2	N=512 tarefas; $P \in \{8, 16, 32\}$
Cenário 3	N=1024 tarefas; $P \in \{8, 16, 32\}$
Cenário 4	N=2048 tarefas; $P \in \{8, 16, 32\}$
Cenário 5	N=4096 tarefas; $P \in \{8, 16, 32\}$

Nota sobre o idioma do repositório. O repositório, gráficos e tabelas associados a este trabalho (GitHub) estão em língua inglesa para facilitar (i) a reprodutibilidade e a revisão por pesquisadores internacionais, (ii) a padronização de nomes, comentários e *docstrings* em alinhamento com a literatura (HEFT, PEFT, IHEFT, IPEFT), e (iii) a colaboração por meio de *issues*/*pull requests* de contribuidores externos. A monografia permanece em português conforme as normas institucionais.

Todos os DAGs e matrizes de custo seguem as propriedades "PEFT-like" (camadas, graus, pesos e heterogeneidade) e são gerados com **semente reprodutível**. O código e *scripts* para execução encontram-se em repositório *open-source* com **licença MIT**; o passo a passo de reprodução e detalhes de implementação estão no **Apêndice A**.

4.1 Resultados por cenário

Abaixo reportamos e discutimos os resultados para cada par (N, P). Para cada cenário, mostramos *um* DAG ilustrativo no início (apenas para dar noção de dimensão) e enfatizamos o painel de métricas definidas em 2.5 — MS, balanceamento de carga $\bar{U}$,

tempo de espera $\overline{\mathrm{WT}}$ e energia E_{total}. Os gráficos de Gantt (um por algoritmo) são usados para interpretar padrões de ocupação, lacunas e criticidade.

4.2 Cenário 1: $N = 256$ tarefas

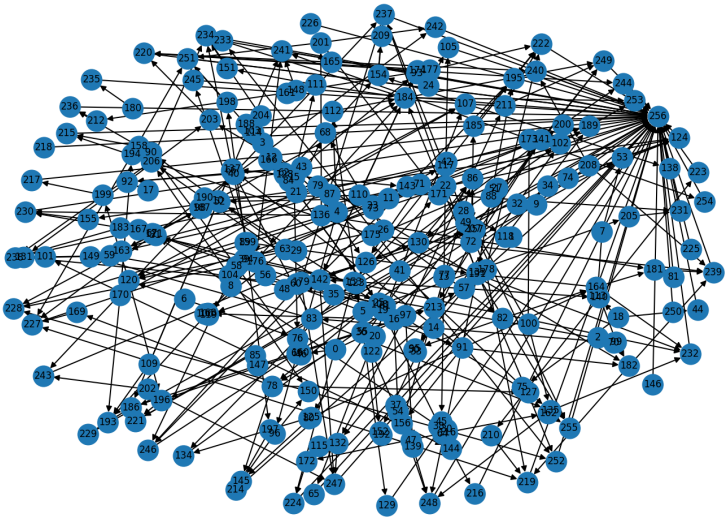


Figura 3 – DAG ilustrativo do cenário 1 (N=256).

O DAG ilustrativo do cenário 1 está mostrado na Figura 3.

4.2.1 Configuração com $P = 8$

4.2.1.1 Análise P=8

Em Figura 4, o **PEFT** apresenta o menor MS ($\approx$ 182), seguido de **IPEFT** ($\approx$ 193), **HEFT** ($\approx$ 205) e **HEFT-LA** ($\approx$ 206). O **IHEFT** fica bem atrás ($\approx$ 314) e o **DLS** é o pior ($\approx$ 678). Em termos relativos ao melhor, os acréscimos de MS são: IPEFT +6%, HEFT +12,6%, HEFT-LA +13,2%, IHEFT +72,5% e DLS +272,5%.

O **balanceamento** $\bar{U}$ (razão MS/busy médio, ideal $\approx$ 1) evidencia um *trade-off*: **DLS** fica praticamente perfeito ($\approx$ 1,03), mas com MS muito alto; já **PEFT**, **IPEFT**, **HEFT** e **HEFT-LA** exibem $\bar{U} \in [0{,}31, 0{,}34]$, indicando que apenas um subconjunto dos processadores carrega a maior parte do trabalho. Esse padrão é coerente com os Gantts: (i) no Figura 8 e Figura 10, dois PEs dominam a execução até o término, enquanto os demais ficam subutilizados após as fases iniciais; (ii) no Figura 6 e Figura 7, a ocupação é mais espalhada, mas ainda com forte concentração em poucos PEs; (iii) no Figura 5, quase não há ociosidade, porém a alta fragmentação e *buracos* causados por precedências/comunicação alongam significativamente a linha do tempo; (iv) já no Figura 9, observam-se janelas ociosas prolongadas em PEs menos velozes, reforçando o MS elevado desta heurística.

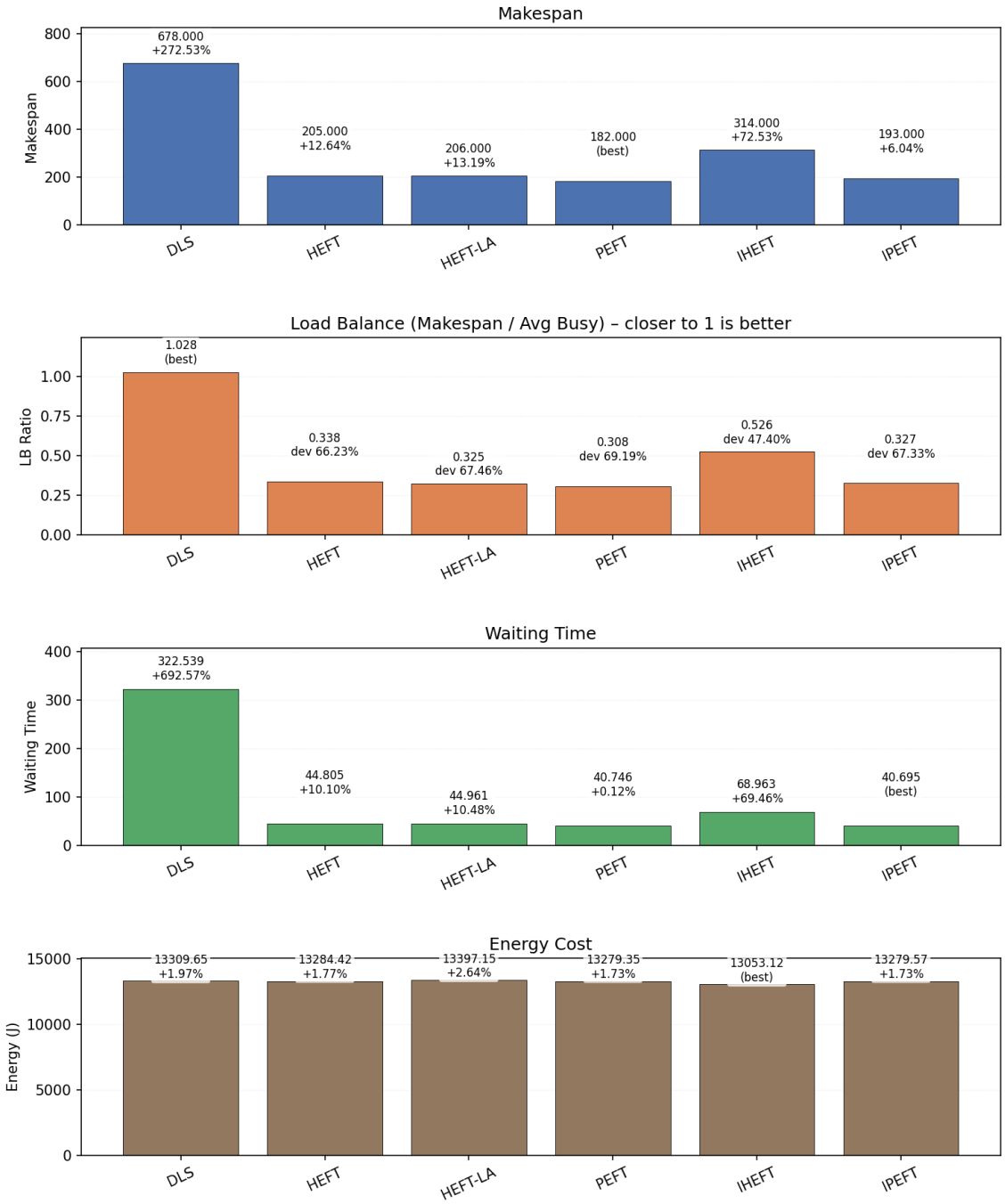


Figura 4 – Métricas (MS, $\bar{U}$, $\overline{\mathrm{WT}}$, E_{total}) – cenário 1, P=8.

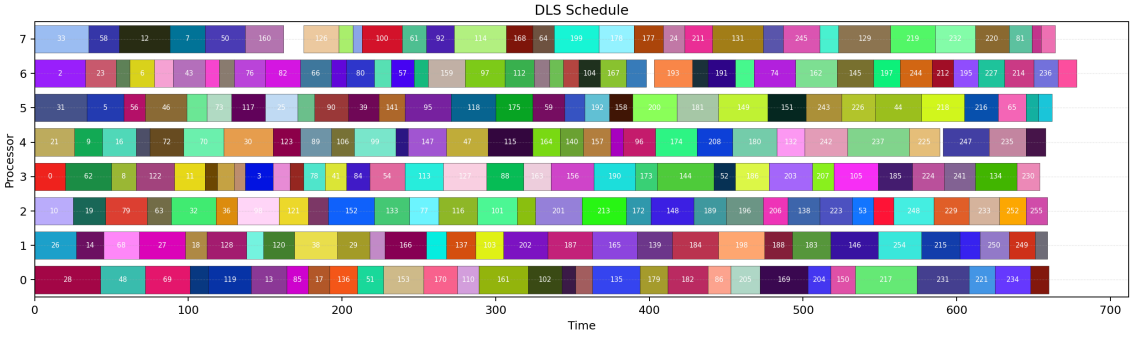


Figura 5 – Gantt – DLS, cenário 1, P=8.

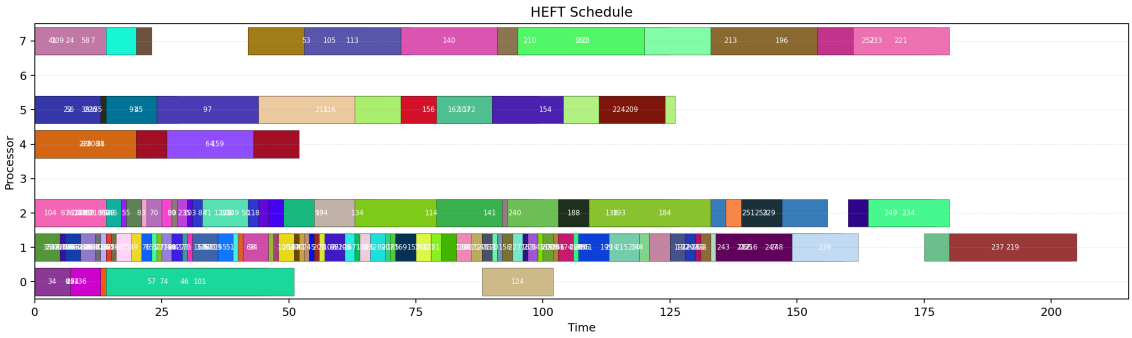


Figura 6 – Gantt – HEFT, cenário 1, P=8.

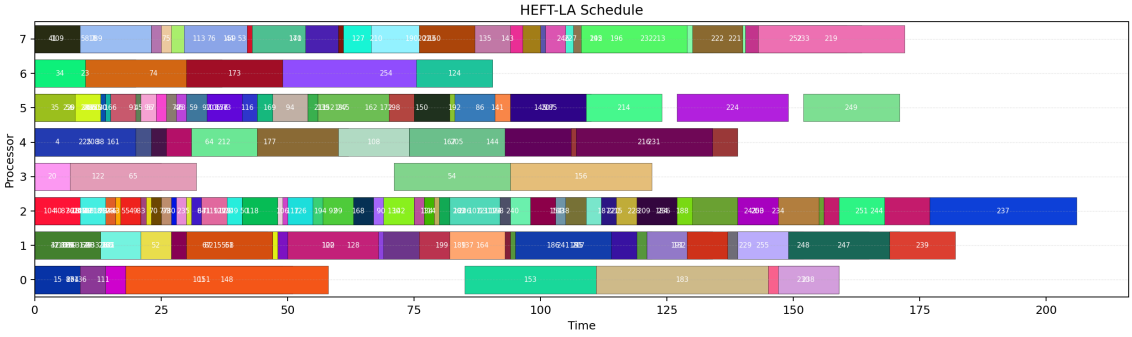


Figura 7 – Gantt – HEFT-LA, cenário 1, P=8.

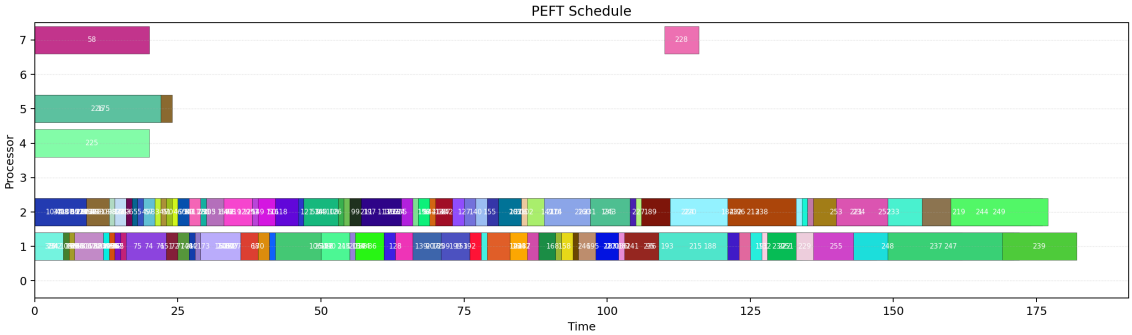


Figura 8 – Gantt – PEFT, cenário 1, P=8.

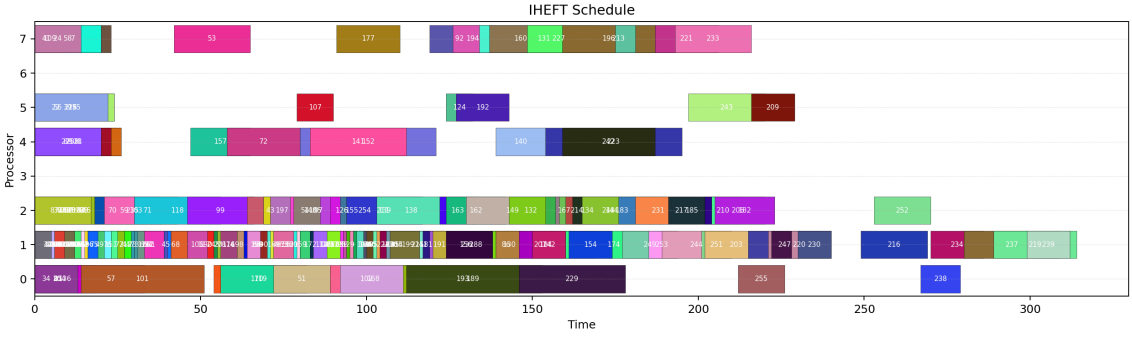


Figura 9 – Gantt – IHEFT, cenário 1, P=8.

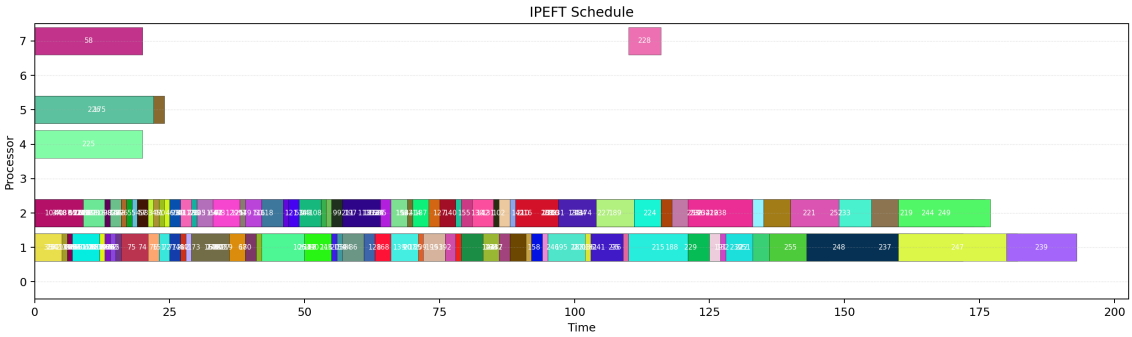


Figura 10 – Gantt – IPEFT, cenário 1, P=8.

No **tempo médio de espera** $\overline{\text{WT}}$, **IPEFT** e **PEFT** lideram (40,7), seguidos de **HEFT** e **HEFT-LA** (45); **IHEFT** eleva a fila (69) e **DLS** é muito pior (323). A leitura dos Gantts confirma: as políticas de inserção/antecipação de {PEFT,IPEFT} reduzem filas e encaixam tarefas críticas nos PEs mais rápidos, enquanto o DLS introduz escalonamento mais miúdo, com esperas acumuladas antes de nós críticos.

Quanto à **energia** E_{total}, as diferenças são pequenas ($\sim 2\%$). O **IHEFT** atinge o menor custo (13053), sugerindo um mapeamento preferencial a PEs mais eficientes energeticamente (ainda que mais lentos), ao preço de um MS 72% maior que o melhor. **PEFT**, **IPEFT** e **HEFT** ficam $\approx 1{,}7{-}1{,}8\%$ acima do mínimo energético, com ganhos expressivos de tempo. Em suma: *para tempo-de-treinamento*, {PEFT,IPEFT} são superiores; *para custo energético estrito*, o IHEFT é ligeiramente melhor.

Do ponto de vista do **perfil de execução**, este caso é predominantemente *compute-bound*: os Gantts de {PEFT,IPEFT} mostram blocos longos, pouca intercalação e baixa migração entre PEs, o que reduz latências de comunicação ao concentrar no(s) PEs mais rápidos. Em contrapartida, o excelente $\bar{U}$ do DLS não se traduz em menor MS porque balancear "tudo com tudo" em hardware heterogêneo amplifica a comunicação e os atrasos de precedência no caminho crítico.

4.2.1.2 Síntese P=8

PEFT entrega o melhor MS (182), com **IPEFT** muito próximo (+6%) e **HEFT** e **HEFT-LA** logo atrás ($+12{-}13\%$). **IHEFT** minimiza E_{total} porém com tempo bem maior, e **DLS** é dominado em todas as métricas críticas de tempo. O padrão de concentração de carga em poucos PEs (baixa $\bar{U}$) nos melhores tempos sugere que, neste DAG, explorar *apenas as GPUs mais rápidas* é mais efetivo do que tentar balancear globalmente — insight relevante para treinamento de LLMs em clusters com forte heterogeneidade.

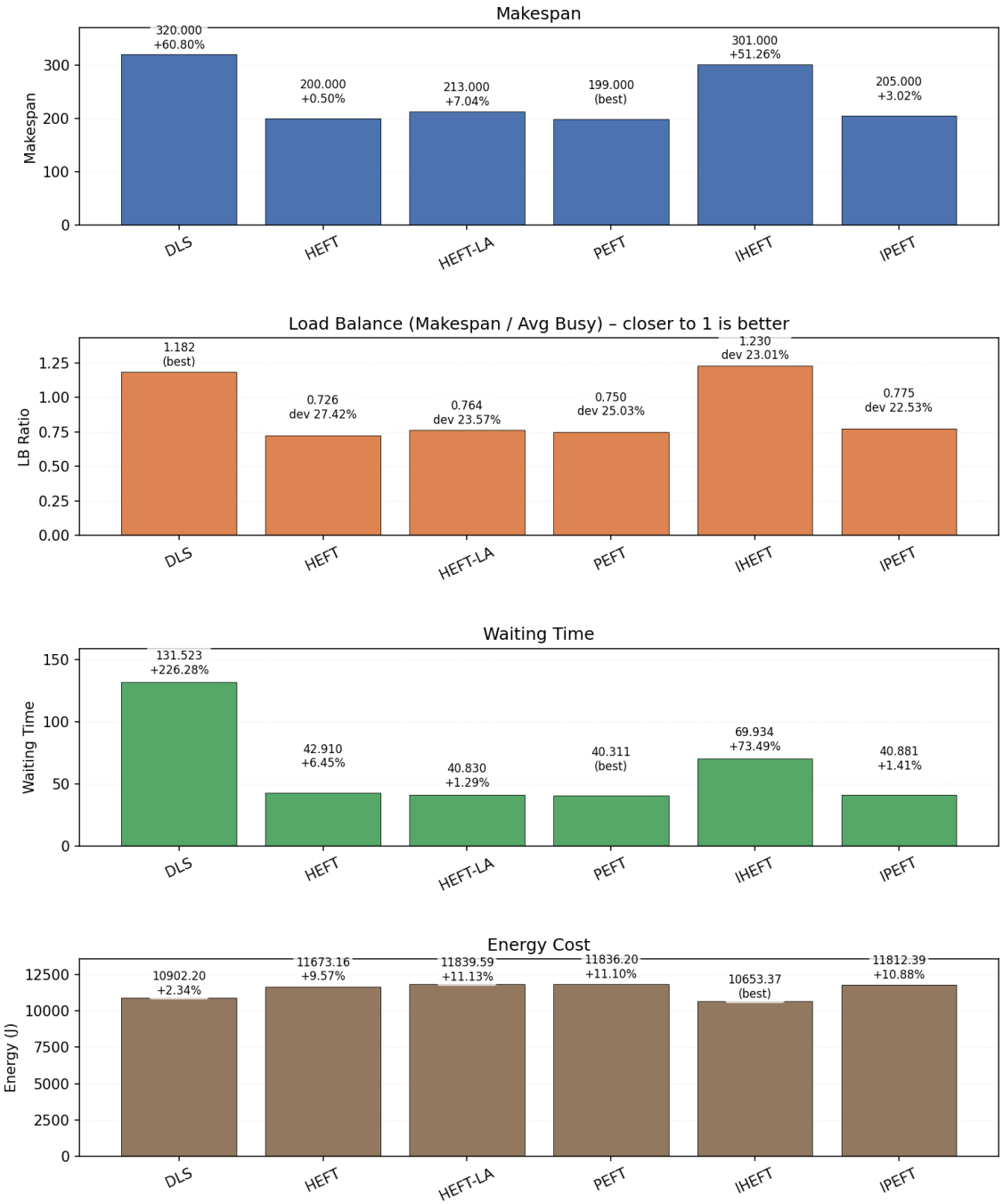


Figura 11 – Métricas (MS, $\bar{U}$, $\overline{\mathrm{WT}}$, E_{total}) – cenário 1, P=16.

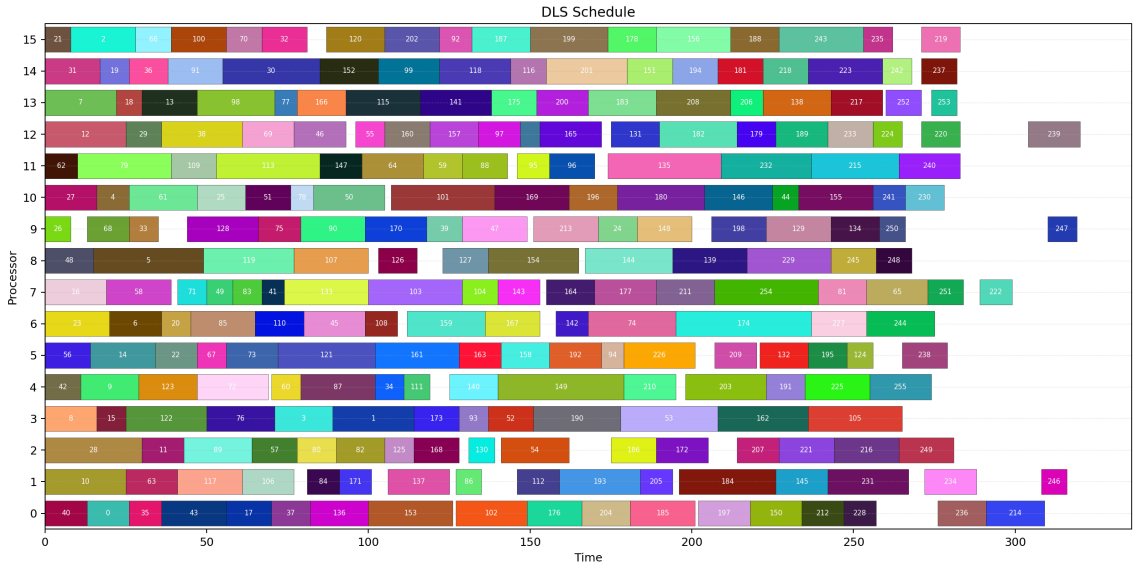


Figura 12 – Gantt – DLS, cenário 1, P=16.

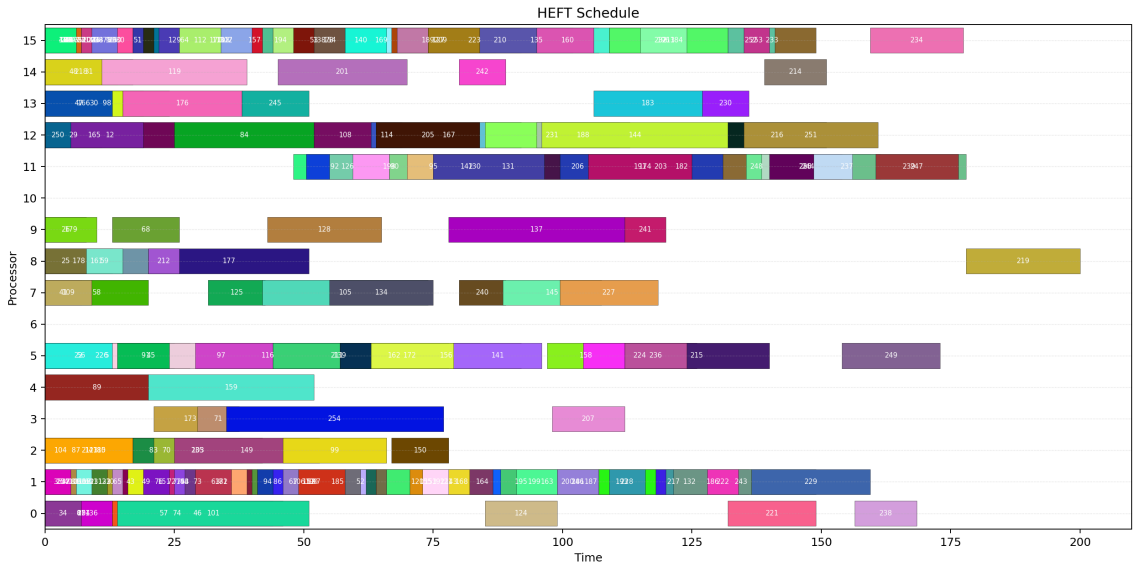


Figura 13 – Gantt – HEFT, cenário 1, P=16.

4.2.2 Configuração com $P = 16$

4.2.2.1 Análise P=16

Em Figura 11, o ranking por MS mantém **PEFT** na liderança (**199**), seguido por **HEFT** (200, +0,5%), **IPEFT** (205, +3,0%) e **HEFT-LA** (213, +7,0%). **IHEFT** (301, +51,3%) e **DLS** (320, +60,8%) ficam distantes. O **balanceamento** $\bar{U}$ melhora substancialmente para as heurísticas "rápidas" (HEFT/HEFT-LA/PEFT/IPEFT $\approx$ 0,73−0,78, com desvios 22−27%), mas ainda indica *concentração* de carga em poucos PEs. O **DLS**, como antes, apresenta o $\bar{U}$ mais próximo de 1 (1,182), porém com MS muito superior ao dos demais; o **IHEFT** tem $\bar{U}$ ainda mais afastado do ideal (1,230), coerente com lacunas visíveis no Figura 16.

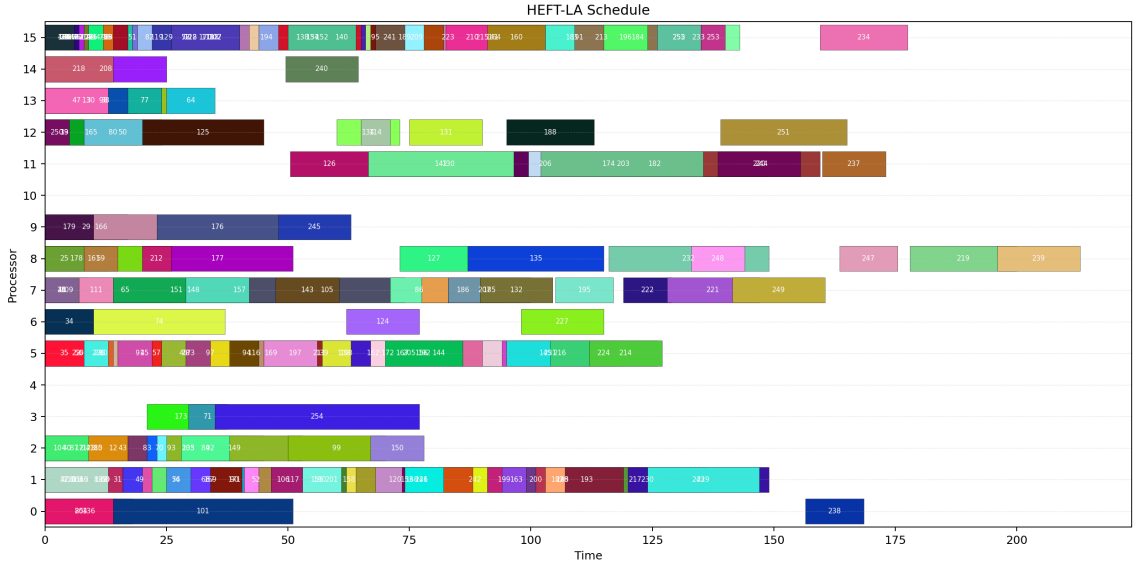


Figura 14 – Gantt – HEFT-LA, cenário 1, P=16.

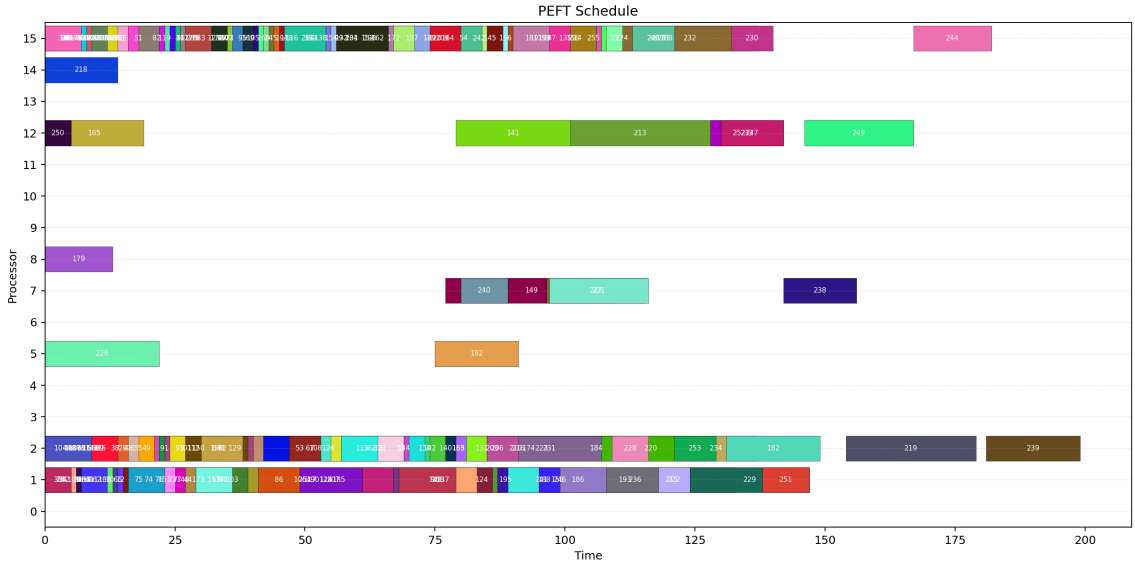


Figura 15 – Gantt – PEFT, cenário 1, P=16.

Nos **Gantts**, Figura 15 e Figura 17 mostram que {PEFT,IPEFT} continuam priorizando os PEs mais rápidos (faixas inferiores), deixando tarefas longas e poucas para PEs adicionais; isso reduz filas e mantém $\overline{\text{WT}}$ baixo (**PEFT** é o melhor com $\approx$ 40,31, **IPEFT** $\approx$ 40,88; **HEFT-LA** $\approx$ 40,83; **HEFT** $\approx$ 42,91), ao passo que **IHEFT** e principalmente **DLS** acumulam esperas (69,93 e 131,52, respectivamente). O Figura 12 evidencia forte fragmentação com muitas trocas entre PEs; já Figura 13 e Figura 14 distribuem melhor a ocupação, mas ainda deixam "ilhas" de ociosidade após o meio da execução, refletindo gargalos de precedência e comunicação.

Em **energia** E_{total}, as variações permanecem modestas, porém o **IHEFT** volta a ser o mais econômico (10653), seguido por **DLS** (10902). As heurísticas mais rápidas

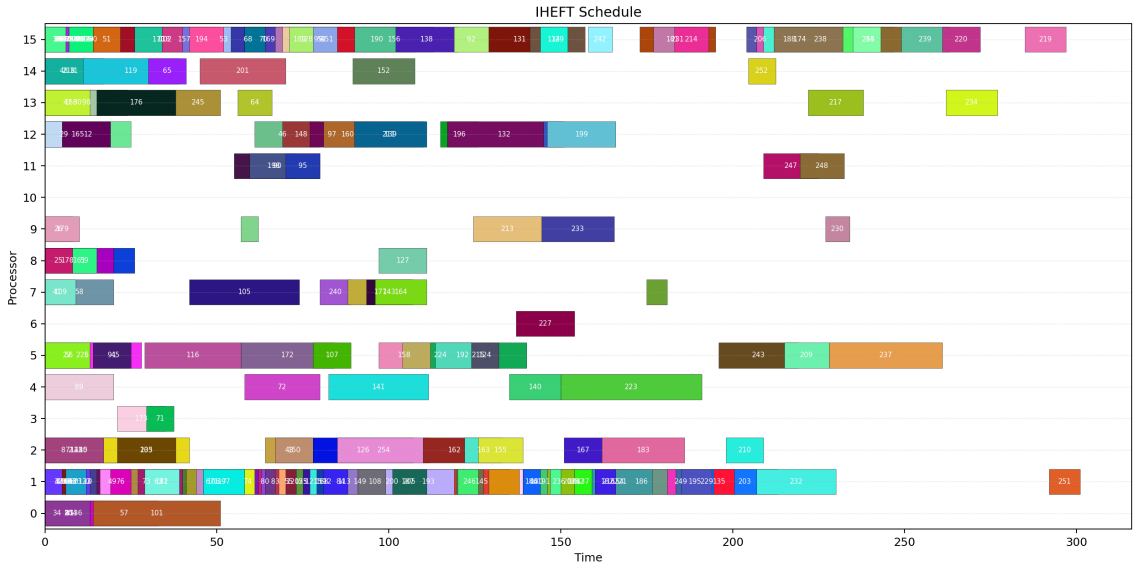


Figura 16 – Gantt – IHEFT, cenário 1, P=16.

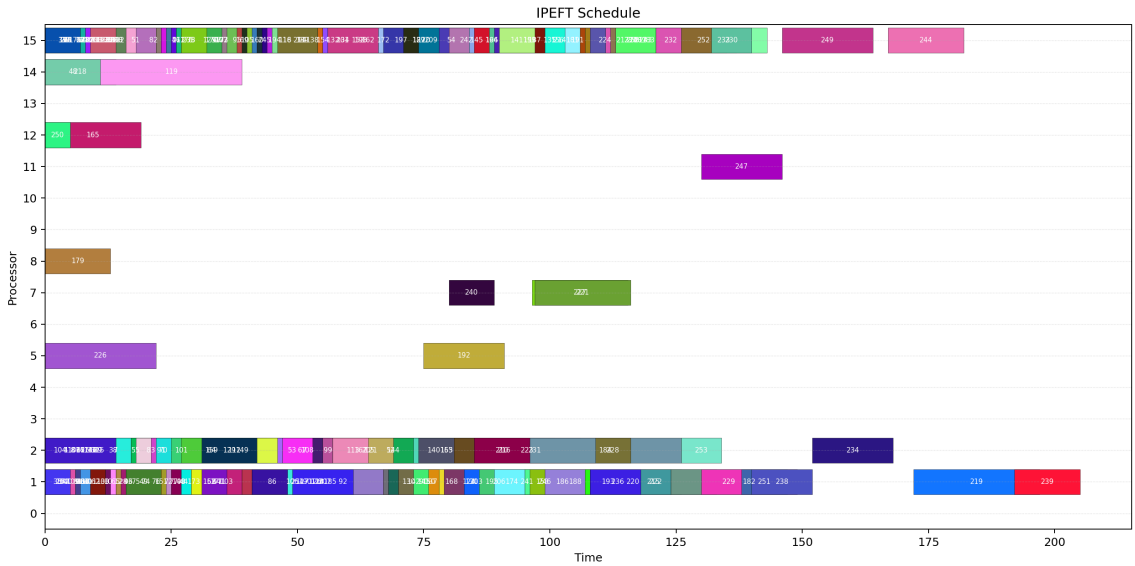


Figura 17 – Gantt – IPEFT, cenário 1, P=16.

({PEFT,IPEFT,HEFT,HEFT-LA}) ficam $\sim$ 9–11% acima do mínimo, sugerindo que trocam parte da eficiência energética por MS menor — um compromisso frequentemente aceitável no treinamento de LLMs com *orçamento de tempo* rígido.

4.2.2.2 Síntese P=16

(i) **PEFT** mantém o menor MS e a menor $\overline{\text{WT}}$; (ii) **HEFT** torna-se o segundo melhor em MS (supera **IPEFT** por margem pequena); (iii) **IHEFT** segue como alternativa de **menor energia**, com penalidade significativa de tempo; (iv) **DLS** melhora muito com mais PEs, mas segue dominado em MS. Para treinamento de LLMs, os resultados reforçam a estratégia de *concentrar* a maior parte do trabalho nos dispositivos mais velozes,

aceitando um $\bar{U}$ menor quando o objetivo principal é tempo de conclusão.

4.2.3 Configuração com $P = 32$

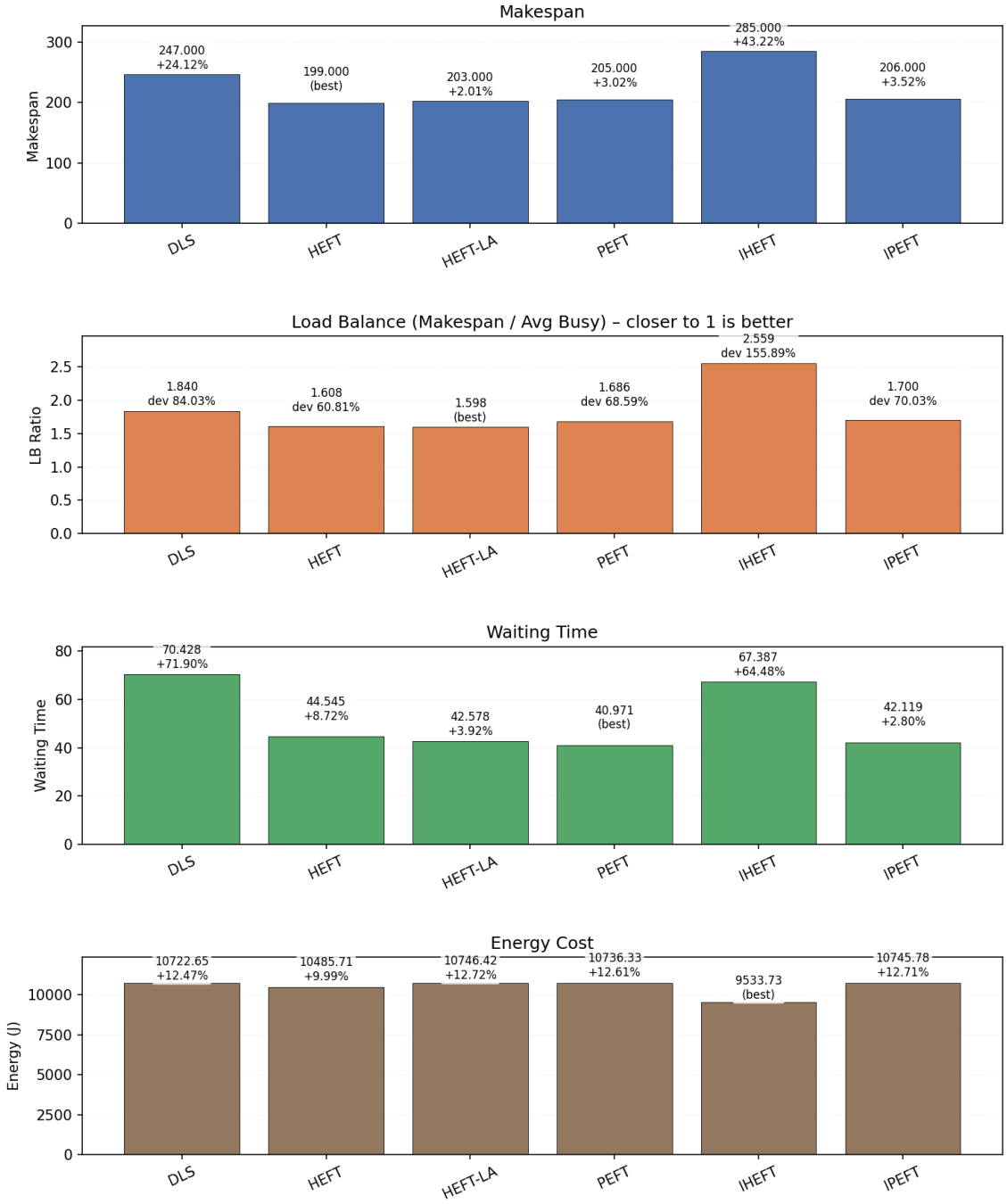


Figura 18 – Métricas (MS, $\bar{U}$, $\overline{\mathrm{WT}}$, E_{total}) – cenário 1, P=32.

4.2.3.1 Análise P=32

O ranking por MS muda: **HEFT** passa a liderar com **199**, seguido de **HEFT-LA** (203, +2,01%), **PEFT** (205, +3,02%) e **IPEFT** (206, +3,52%); **DLS** (247, +24,12%) e **IHEFT** (285, +43,22%) ficam atrás (Figura 18). O **balanceamento** (razão próxima de



Figura 19 – Gantt – DLS, cenário 1, P=32.

1 é melhor) evidencia *subutilização generalizada*: todos os métodos ficam acima de 1,5; o mais próximo do ideal é **HEFT-LA** (1,598), seguido de **HEFT** (1,608). **IHEFT** é o pior (2,559), coerente com grandes lacunas no Figura 23.

Nos **Gantts**, a saturação de paralelismo é clara. Em Figura 19, o DLS pulveriza tarefas em muitos PEs com forte fragmentação e longas "pontas" de ociosidade — daí o MS elevado. Em Figura 20 e Figura 21, as execuções se concentram em subconjuntos de PEs, com blocos longos e menor fragmentação; isso reduz dependências no caminho crítico e explica o MS superior de HEFT/HEFT-LA. Já Figura 22 e Figura 24 mantêm a estratégia de "agrupar" em PEs mais rápidos, o que minimiza $\overline{\mathrm{WT}}$ mas não converte em melhor MS sob muitos PEs; por fim, o Figura 23 evidencia blocos grandes em PEs mais lentos e tarefas residuais que arrastam a cauda.

No **tempo médio de espera**, **PEFT** é o melhor (40,971), com **IPEFT** e **HEFT-LA** muito próximos (42,119 e 42,578); **HEFT** é um pouco mais alto (44,545). **DLS** (70,428) e **IHEFT** (67,387) voltam a concentrar filas e reentrâncias. Em **energia**, o



Figura 20 – Gantt – HEFT, cenário 1, P=32.

IHEFT é novamente o mais econômico (9533,73), 10–13% abaixo dos demais; o preço é um MS 43% maior que o melhor.

4.2.3.2 Síntese P=32

(i) **HEFT** torna-se o melhor em MS e, junto com **HEFT-LA**, apresenta o melhor $\bar{U}$ entre os métodos; (ii) **PEFT** e **IPEFT** mantêm $\overline{\text{WT}}$ mínimo, mas não extraem ganhos de MS com P=32; (iii) **IHEFT** segue como a opção **mais eficiente energeticamente**, com penalidade considerável em tempo; (iv) O conjunto de resultados 8→16→32 indica **saturação de largura efetiva do DAG** perto de $P \approx 16$: acima disso, o custo de comunicação e as dependências limitam o ganho de paralelismo — ponto crucial para treino de LLMs em clusters heterogêneos.

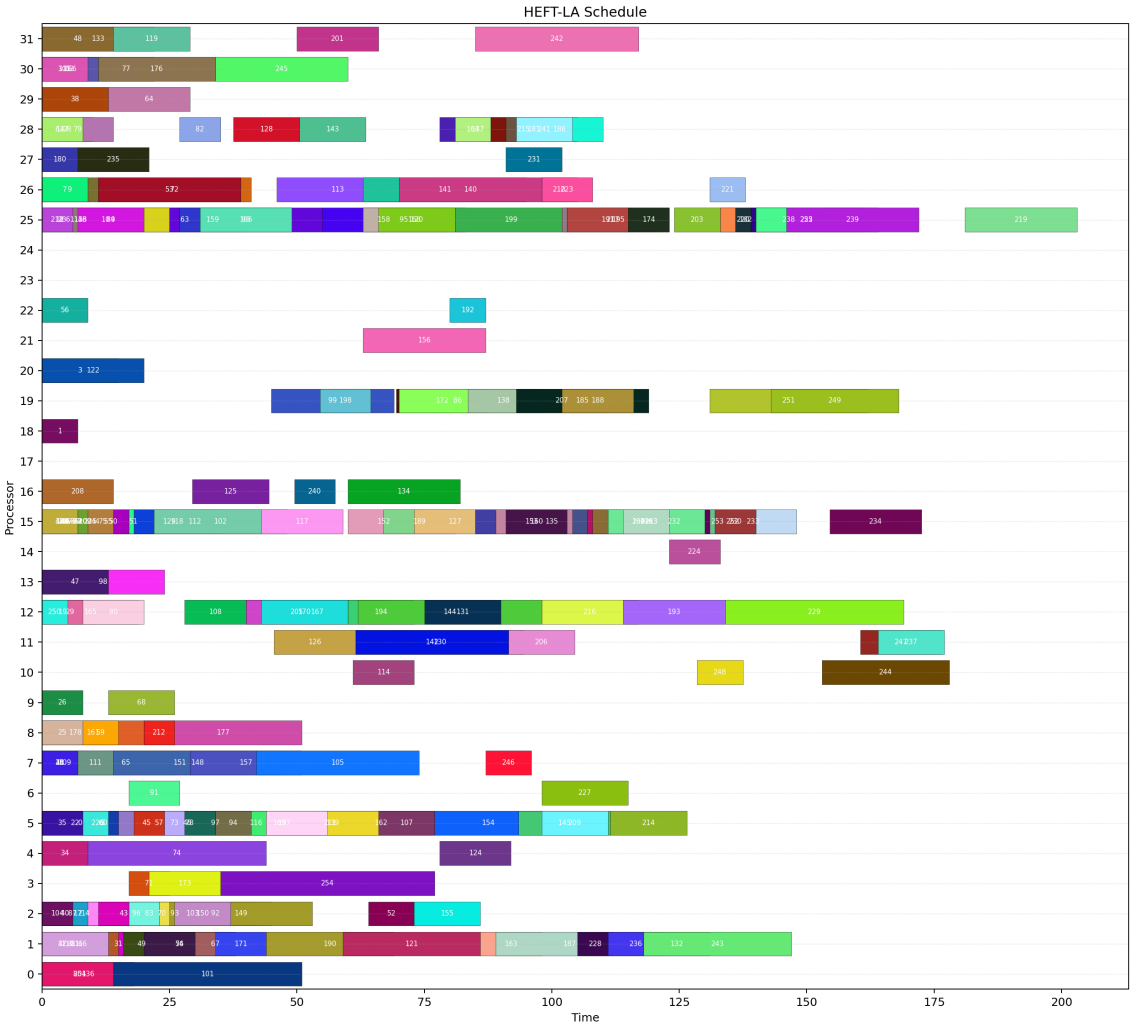


Figura 21 – Gantt – HEFT-LA, cenário 1, P=32.

4.2.4 Síntese do cenário 1 (N=256)

Panorama geral. Nos três níveis de paralelismo, as heurísticas **PEFT** e **IPEFT** entregam os menores $\overline{\text{WT}}$ em todas as configurações (Figura 25c) e figuram entre as melhores em MS (Figura 25a); **HEFT** empata/ultrapassa em MS quando P=32. O **IHEFT** é consistentemente o mais *eficiente energeticamente* (Figura 25d), porém com MS substancialmente maior. O **DLS** melhora bastante ao aumentar P, mas permanece dominado em MS e $\overline{\text{WT}}$.

Saturação de paralelismo. A curva de MS evidencia *largura efetiva* do DAG próxima de $P \approx 16$: de 16 $\rightarrow$ 32 não há queda de MS para os melhores métodos (PEFT/IPEFT/HEFT), e em alguns casos há ligeiro *aumento* por overhead de comunicação/sincronização. Isso se alinha ao crescimento da razão de balanceamento (Figura 25b) — todas as heurísticas se afastam de 1 quando P cresce, indicando maior ociosidade/fragmentação sem ganho de caminho crítico.

Trade-offs tempo vs energia. Enquanto **PEFT** e **IPEFT** minimizam $\overline{\text{WT}}$ e competem

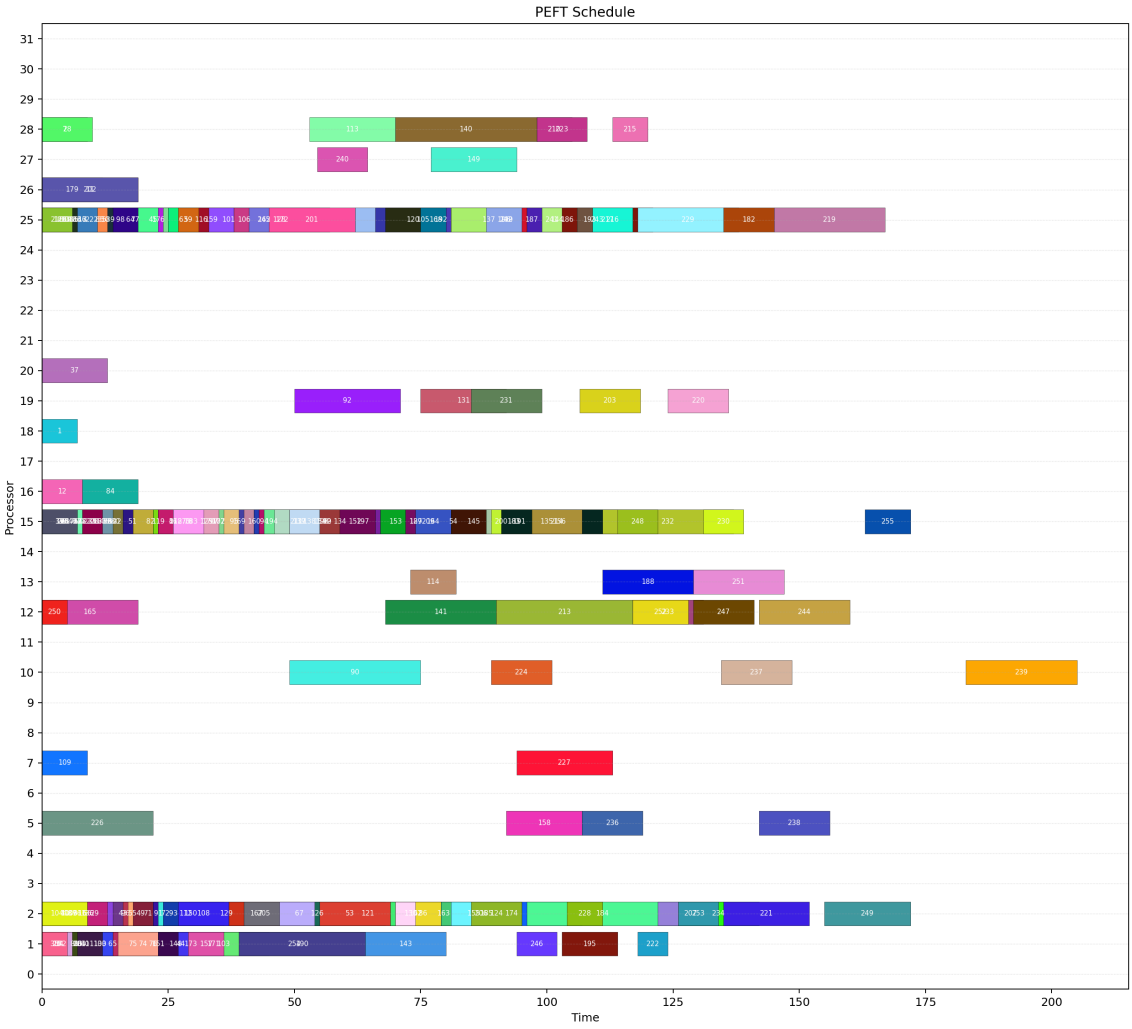


Figura 22 – Gantt – PEFT, cenário 1, P=32.

pelo menor MS (especialmente para P=8 e P=16), o **IHEFT** reduz E_{total} de forma consistente (queda mais acentuada de P=8 → 32), ao preço de MS +40%–+70% acima do melhor — um compromisso relevante quando o objetivo é operar dentro de *orçamentos energéticos* estritos.

Conclusão do cenário 1. Para N=256, o melhor MS ocorre com **PEFT** (P=8, 16) e **HEFT** (P=32); **IPEFT** permanece muito próximo. **PEFT** e **IPEFT** dominam em $\overline{\text{WT}}$, enquanto **IHEFT** domina em E_{total}. O comportamento conjunto confirma um regime majoritariamente *compute-bound* com gargalos de precedência: mais PEs não implica menor MS — a política de mapeamento e a escolha dos PEs *certos* são decisivas.

4.3 Cenário 2: $N = 512$ tarefas

O DAG ilustrativo do cenário 2 está mostrado na Figura 72.



Figura 23 – Gantt – IHEFT, cenário 1, P=32.

4.3.1 Configuração com $P = 8$

4.3.1.1 Análise P=8

Em Figura 27, **PEFT** e **IPEFT** empatam com o menor MS (**162**), seguidos por **HEFT** (166, +2,47%). **HEFT-LA** piora sensivelmente (212, +30,86%) e **IHEFT** fica bem atrás (262, +61,73%). **DLS** é dominado (1308, +707%).

O **balanceamento** $\bar{U}$ (ideal ≈ 1) evidencia o mesmo *trade-off* observado no cenário anterior: **DLS** é o mais "equilibrado" (1,008), mas entrega o pior MS; já os algoritmos **PEFT**, **IPEFT**, **HEFT** e **HEFT-LA** exibem $\bar{U} \in [0{,}144, 0{,}169]$ com altos desvios ($\approx 83{-}86\%$), sugerindo que um subconjunto de PEs absorve a porção crítica do trabalho.

Os **Gantts** corroboram: em Figura 31 e Figura 33 a execução se concentra em poucos PEs com *blocos longos*, minimizando migrações e comunicação ao longo do caminho crítico. Em Figura 29 o padrão é similar, porém com pequenas "ilhas" tardias que explicam a leve perda para PEFT/IPEFT. O Figura 30 espalha mais e cria caudas — maior fragmentação $\Rightarrow$ MS maior. O Figura 32 revela grandes segmentos fixos em um

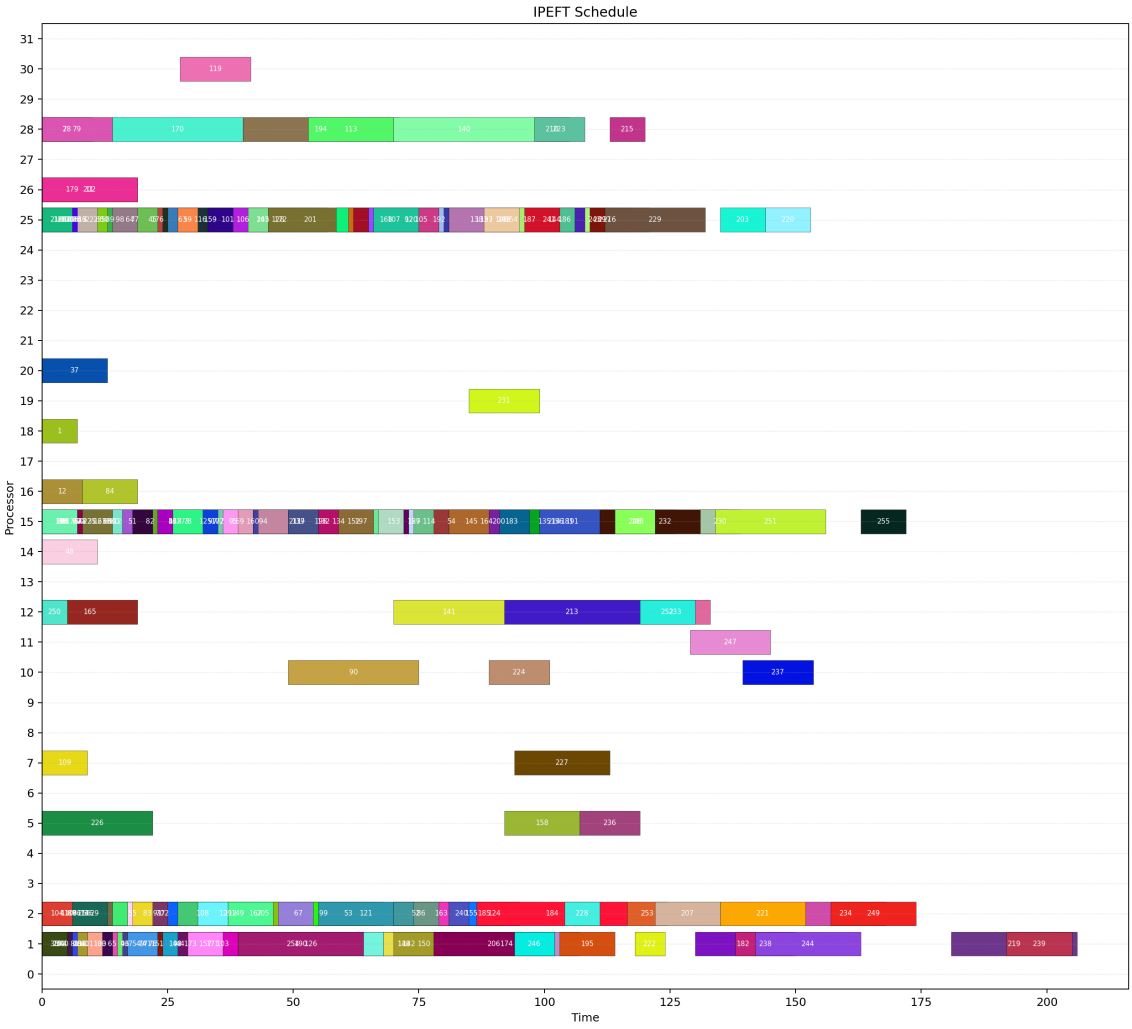


Figura 24 – Gantt – IPEFT, cenário 1, P=32.

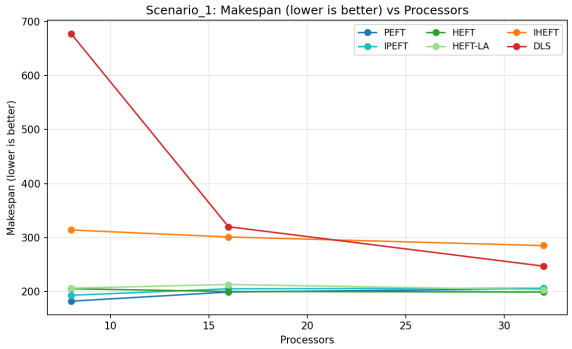
mesmo PE e ociosidade nos demais; o Figura 28 é fortemente fragmentado e estendido no tempo.

No **tempo médio de espera** $\overline{\text{WT}}$, **HEFT** é o melhor (35,272), com **PEFT** (35,447) e **IPEFT** (35,498) essencialmente empatados ($< 1\%$ de diferença). **HEFT-LA** (45,834) e **IHEFT** (49,771) apresentam filas maiores; **DLS** é muito pior (667).

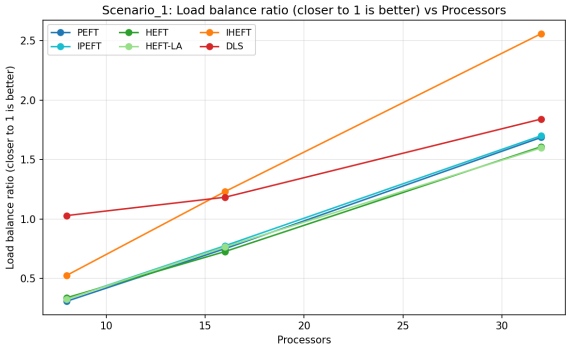
Em **energia** E_{total}, **IHEFT** volta a ser o mais econômico (24828), enquanto **HEFT** e **PEFT** e **IPEFT** ficam apenas 1,5−1,7% acima do mínimo. **HEFT-LA** é o pior (26187, +5,47%). Assim, para metas de tempo, {PEFT, IPEFT, HEFT} são preferíveis; para orçamento energético estrito, **IHEFT** é a alternativa.

4.3.1.2 Síntese P=8

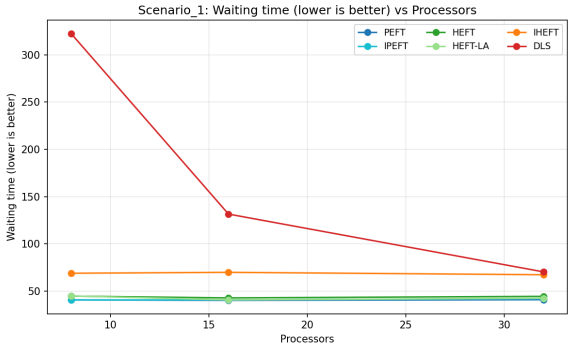
PEFT e **IPEFT** alcançam o menor MS (162) e $\overline{\text{WT}}$ praticamente mínimo; **HEFT** vem muito próximo em ambas as métricas. **IHEFT** minimiza E_{total}, porém com penalidade de MS (+62% vs. melhor). **DLS** é dominado. O DAG de N=512 reforça um regime



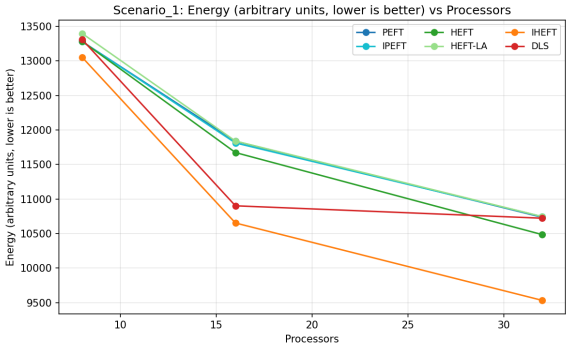
(a) Makespan por número de processadores (↓ melhor).



(b) Razão de balanceamento de carga (≈ 1 é melhor).



(c) Tempo médio de espera $\overline{\mathrm{WT}}$ (↓ melhor).



(d) Energia total E_{total} (↓ melhor).

Figura 25 – Síntese visual do cenário 1: evolução das métricas com $P \in \{8, 16, 32\}$.

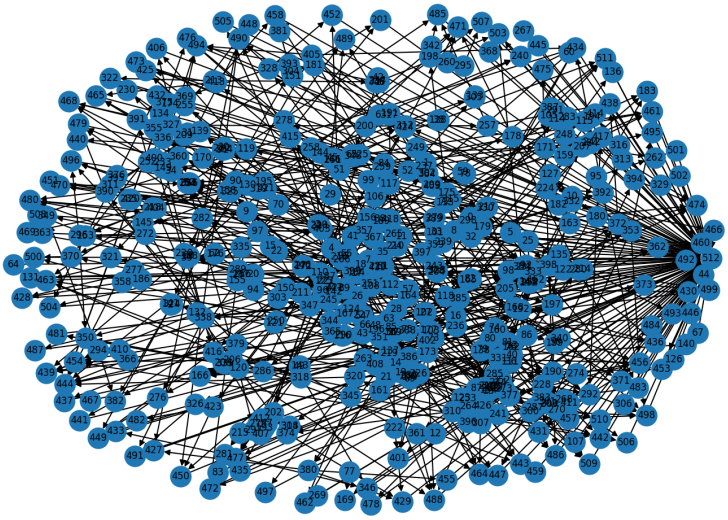


Figura 26 – DAG ilustrativo do cenário 2 (N=512).

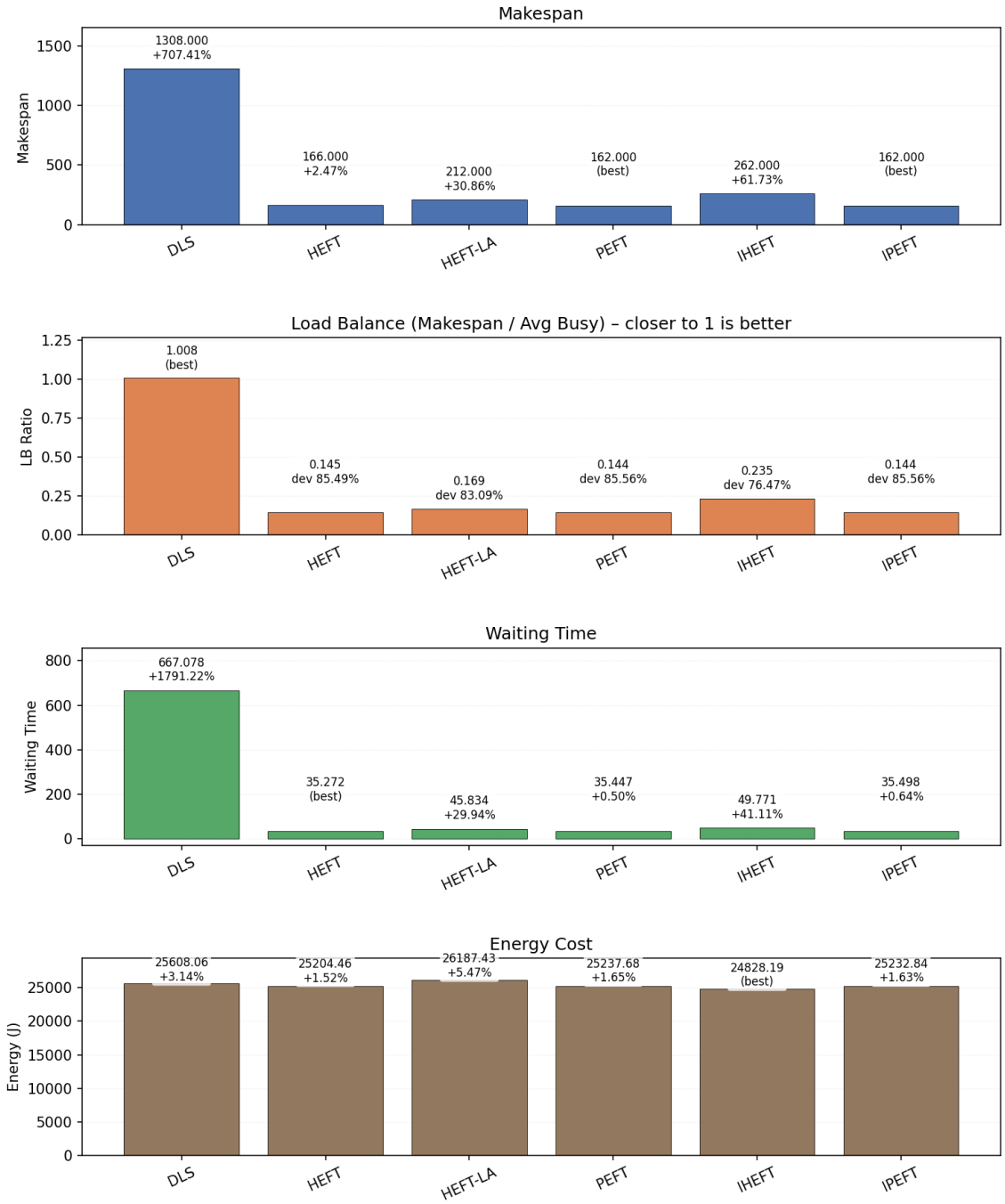


Figura 27 – Métricas (MS, $\bar{U}$, $\overline{\mathrm{WT}}$, E_{total}) – cenário 2, P=8.

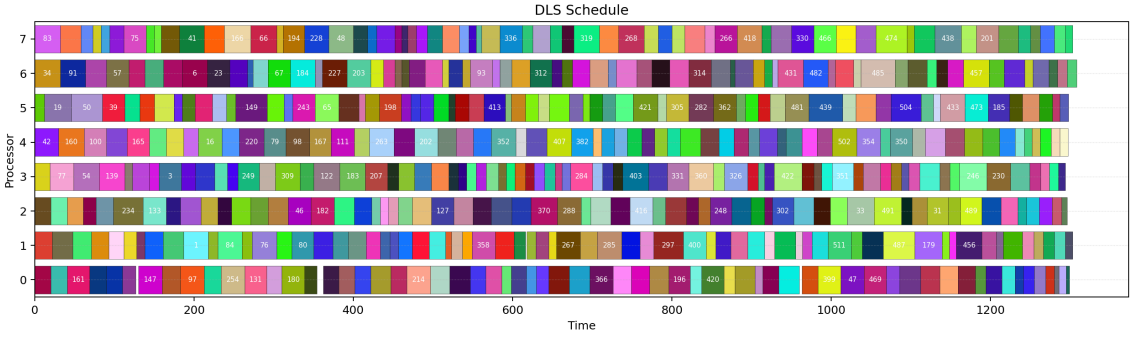


Figura 28 – Gantt – DLS, cenário 2, P=8.

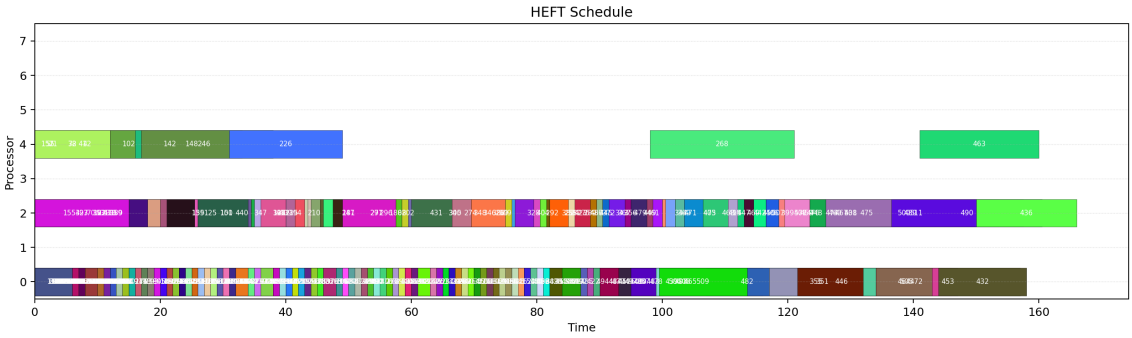


Figura 29 – Gantt – HEFT, cenário 2, P=8.

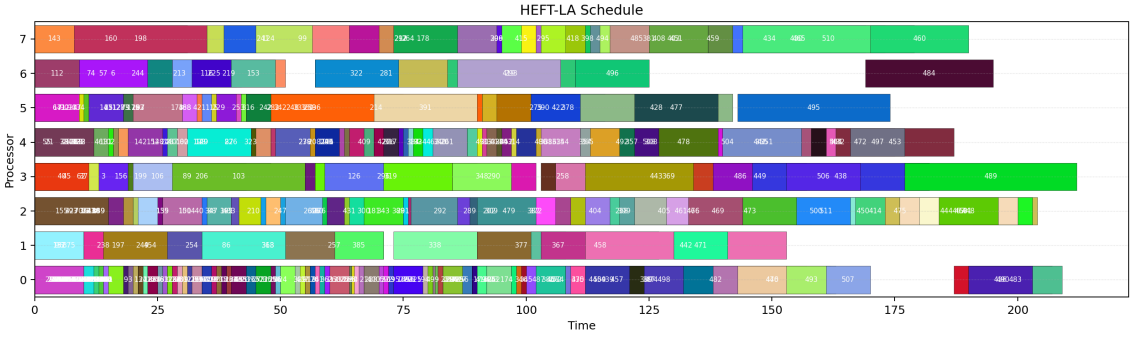


Figura 30 – Gantt – HEFT-LA, cenário 2, P=8.

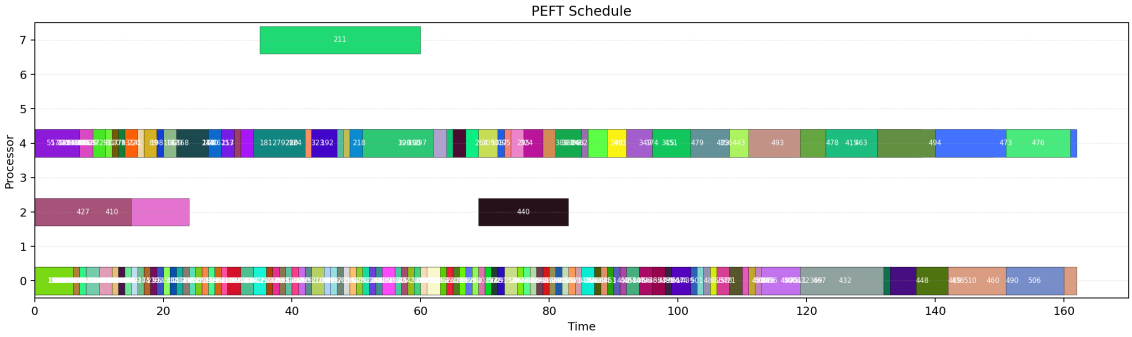


Figura 31 – Gantt – PEFT, cenário 2, P=8.

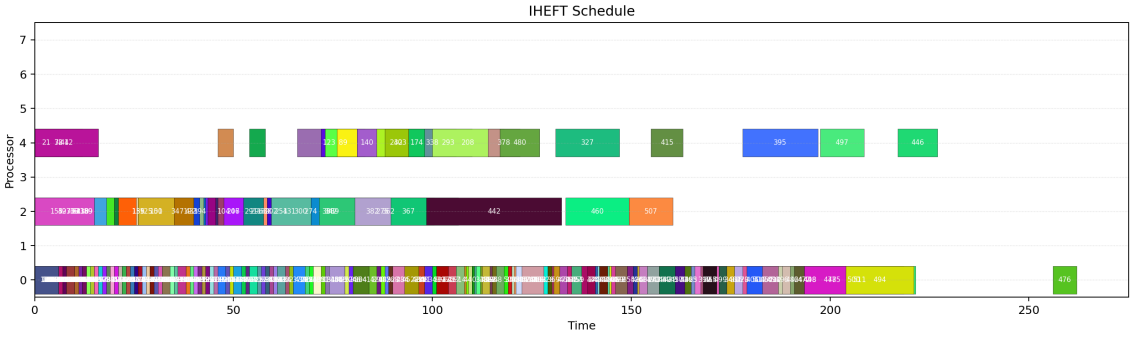


Figura 32 – Gantt – IHEFT, cenário 2, P=8.

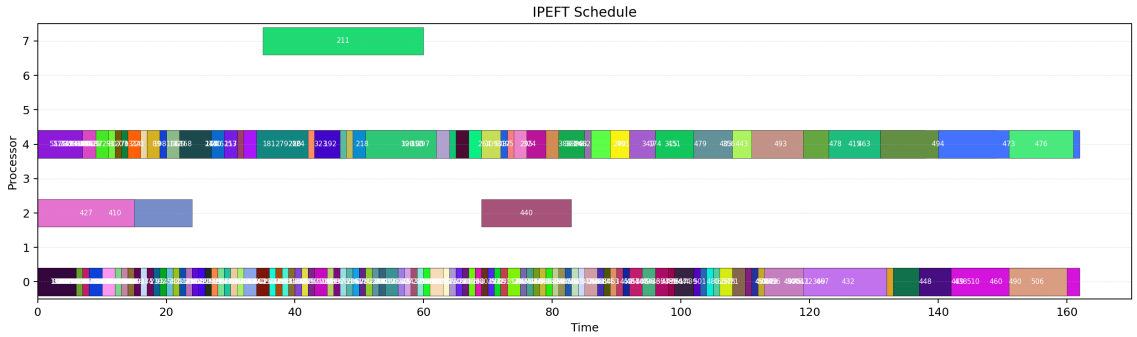


Figura 33 – Gantt – IPEFT, cenário 2, P=8.

compute-bound: concentrar tarefas críticas nas GPUs mais rápidas rende ganhos de tempo substanciais mesmo com $\bar{U}$ global baixo.

4.3.2 Configuração com $P = 16$

4.3.2.1 Análise P=16

Makespan. PEFT e **IPEFT** obtêm o menor MS (**176,5**), com **HEFT** logo atrás (206, +16,71%) e **HEFT-LA** (212, +20,11%). **IHEFT** piora significativamente (345, +95,47%), e o **DLS** permanece dominado (516, +192,35%); cf. Figura 34.

Balanceamento $\bar{U}$. PEFT e **IPEFT** apresentam $\bar{U} \approx 0{,}34$ (desvios $\sim 66\%$), **HEFT** e **HEFT-LA** ficam em $\sim$0,40, **DLS** é quase "perfeito" (1,025), e **IHEFT** $\approx$0,72.

Leitura dos Gantts. Figura 38 e Figura 40: *blocos longos* em poucos PEs, dependências localizadas e latências reduzidas; Figura 36: padrão semelhante com "ilhas" tardias; Figura 37: maior espalhamento e fragmentação; Figura 39: esparsidade e janelas ociosas tardias; Figura 35: mosaico denso e fragmentado com linha do tempo longa.

Tempo de espera. PEFT é o melhor em $\overline{\mathrm{WT}}$ (37,59), seguido por **IPEFT** (37,66); **HEFT** e **HEFT-LA** (40,58–40,92); **IHEFT** (62,03); **DLS** (251,87).

Energia. O **DLS** apresenta o menor E_{total} (19,994), seguido por **IHEFT** (20,663, +3,34%). **HEFT** e **HEFT-LA** e **PEFT** e **IPEFT** ficam +10−12,5% acima.

4.3.2.2 Síntese P=16

PEFT e **IPEFT** mantêm o melhor MS e o menor $\overline{\mathrm{WT}}$; **HEFT** vem logo atrás. **DLS** minimiza a energia neste caso, mas segue pouco competitivo em tempo.

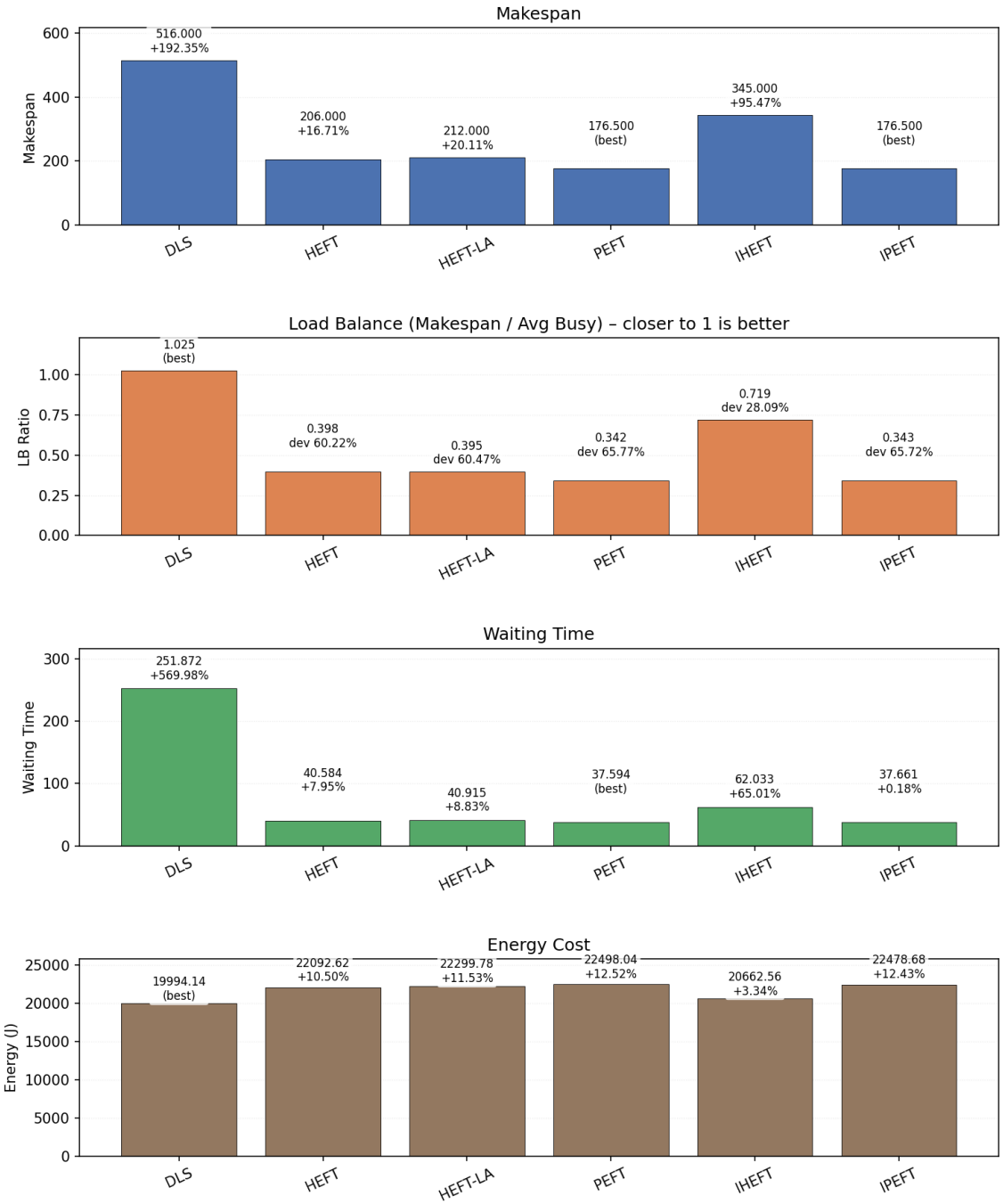


Figura 34 – Métricas (MS, $\bar{U}$, $\overline{\mathrm{WT}}$, E_{total}) – cenário 2, P=16.

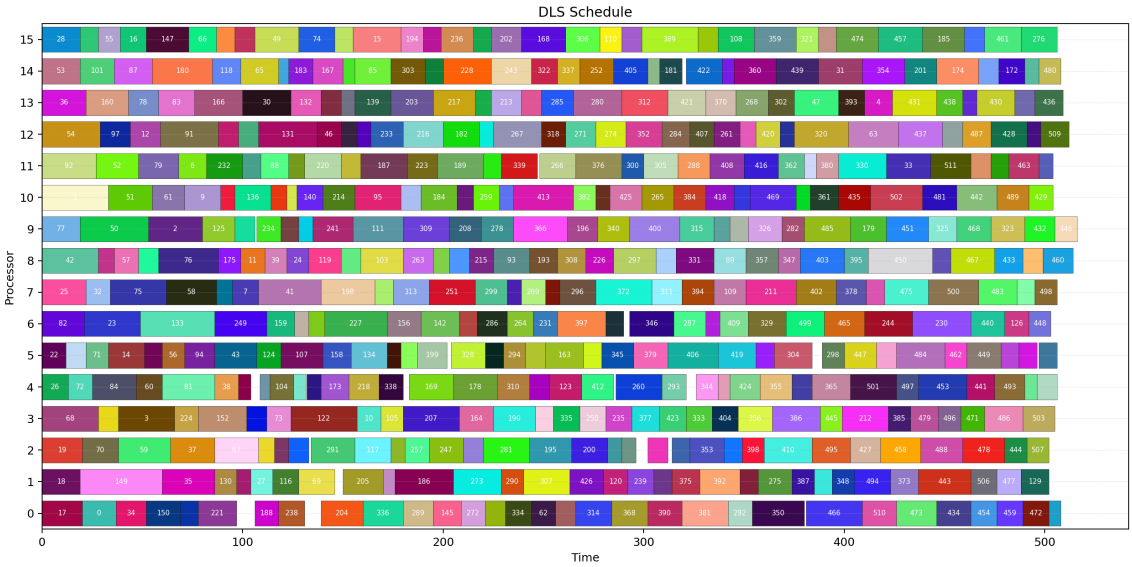


Figura 35 – Gantt – DLS, cenário 2, P=16.

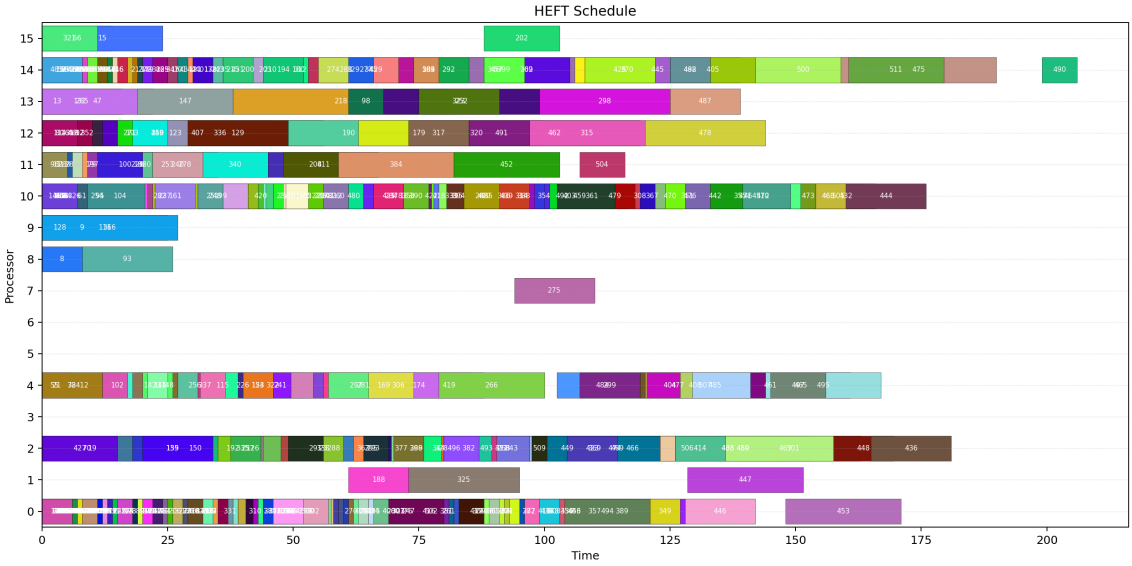


Figura 36 – Gantt – HEFT, cenário 2, P=16.

4.3.3 Configuração com $P = 32$

4.3.3.1 Análise P=32

Makespan. **HEFT** (**184**), **PEFT** e **IPEFT** (191, +3,8%), **HEFT-LA** (197, +7,1%), **DLS** (293, +59,2%), **IHEFT** (315,5, +71,5%); cf. Figura 41.

Balanceamento $\bar{U}$. **HEFT-LA** ≈0,835; **PEFT** e **IPEFT** ∼0,81; **HEFT** ∼0,79; **DLS** 1,197; **IHEFT** 1,501.

Leitura dos Gantts. Figura 45, Figura 47: concentração em subconjunto de PEs rápidos e latências menores; Figura 43: alocação um pouco mais uniforme e caudas discretas (explica o melhor MS); Figura 44: fragmentações suavizadas; Figura 46: blocos grandes

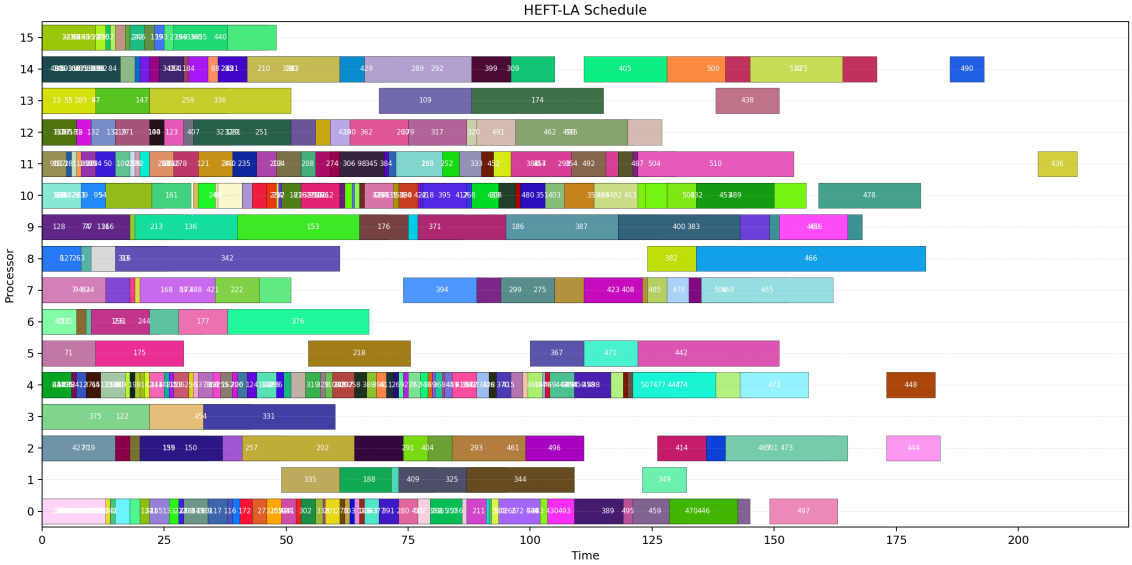


Figura 37 – Gantt – HEFT-LA, cenário 2, P=16.

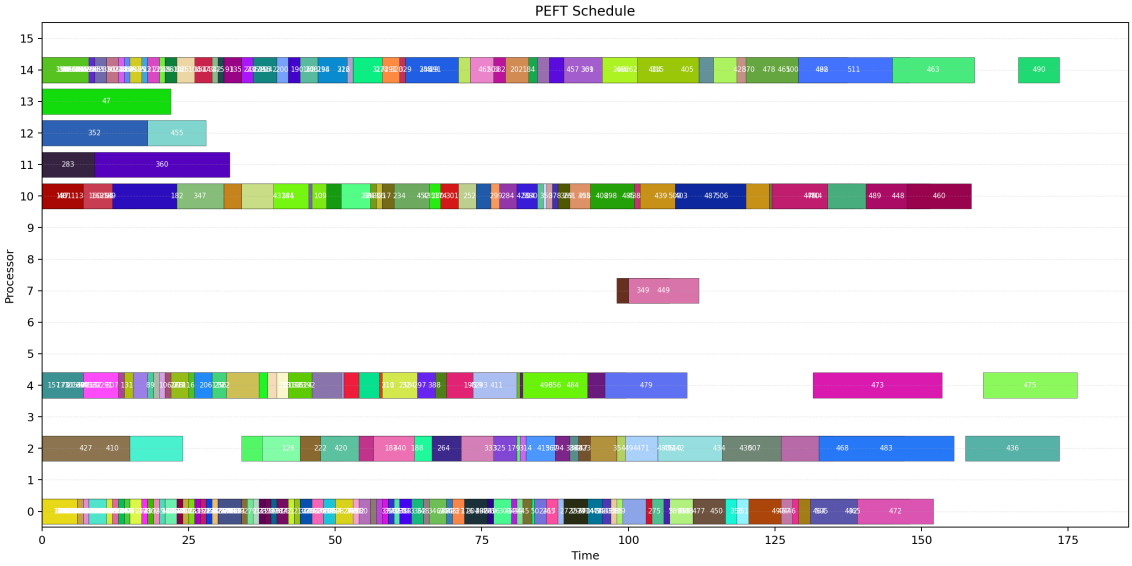


Figura 38 – Gantt – PEFT, cenário 2, P=16.

em PEs menos velozes e tarefas residuais; Figura 42: distribuição intensa e fragmentação que estende a linha do tempo.

Tempo de espera. **PEFT** (38,48) ≈ **IPEFT** (38,49) < **HEFT-LA** (39,45) < **HEFT** (41,30) < **IHEFT** (63,38) < **DLS** (118,64).

Energia. Mínimo em **IHEFT** (18,138); **DLS** +6,6%; **HEFT** e **HEFT-LA** +9−10%; **PEFT** e **IPEFT** +14%.

4.3.3.2 Síntese P=32

HEFT entrega o menor MS (184), com **PEFT** e **IPEFT** muito próximos e **HEFT-LA** na sequência. **PEFT** e **IPEFT** mantêm o melhor $\overline{\text{WT}}$; **IHEFT** minimiza

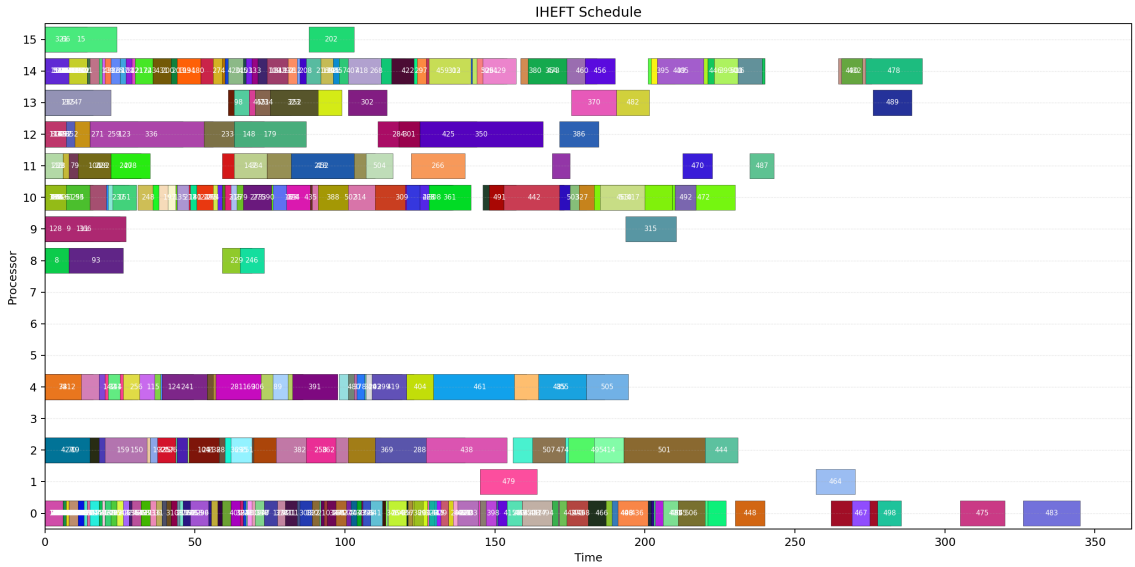


Figura 39 – Gantt – IHEFT, cenário 2, P=16.

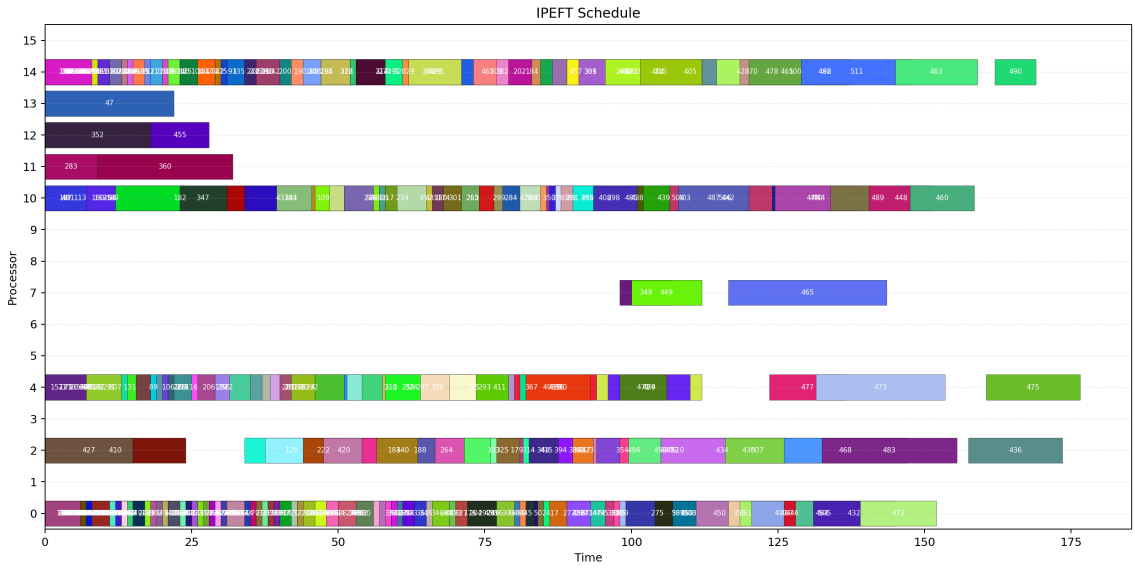


Figura 40 – Gantt – IPEFT, cenário 2, P=16.

energia com forte penalidade em tempo.

4.3.4 Síntese do cenário 2 (N=512)

No conjunto de experimentos do cenário 2, os resultados ao longo de $P \in \{8, 16, 32\}$ preservam o mesmo "perfil" observado no cenário 1, mas com efeitos mais nítidos devido ao maior número de tarefas.

Desempenho em tempo (MS e $\overline{\text{WT}}$). As variantes com *inserção* — **PEFT** e **IPEFT** — mantêm os menores tempos de espera médios e figuram entre os melhores MS em todas as configurações (Figura 48a, Figura 48c). À medida que P cresce, **HEFT** e **HEFT-LA** tornam-se muito competitivos (e, em alguns casos, lideram por pequena mar-

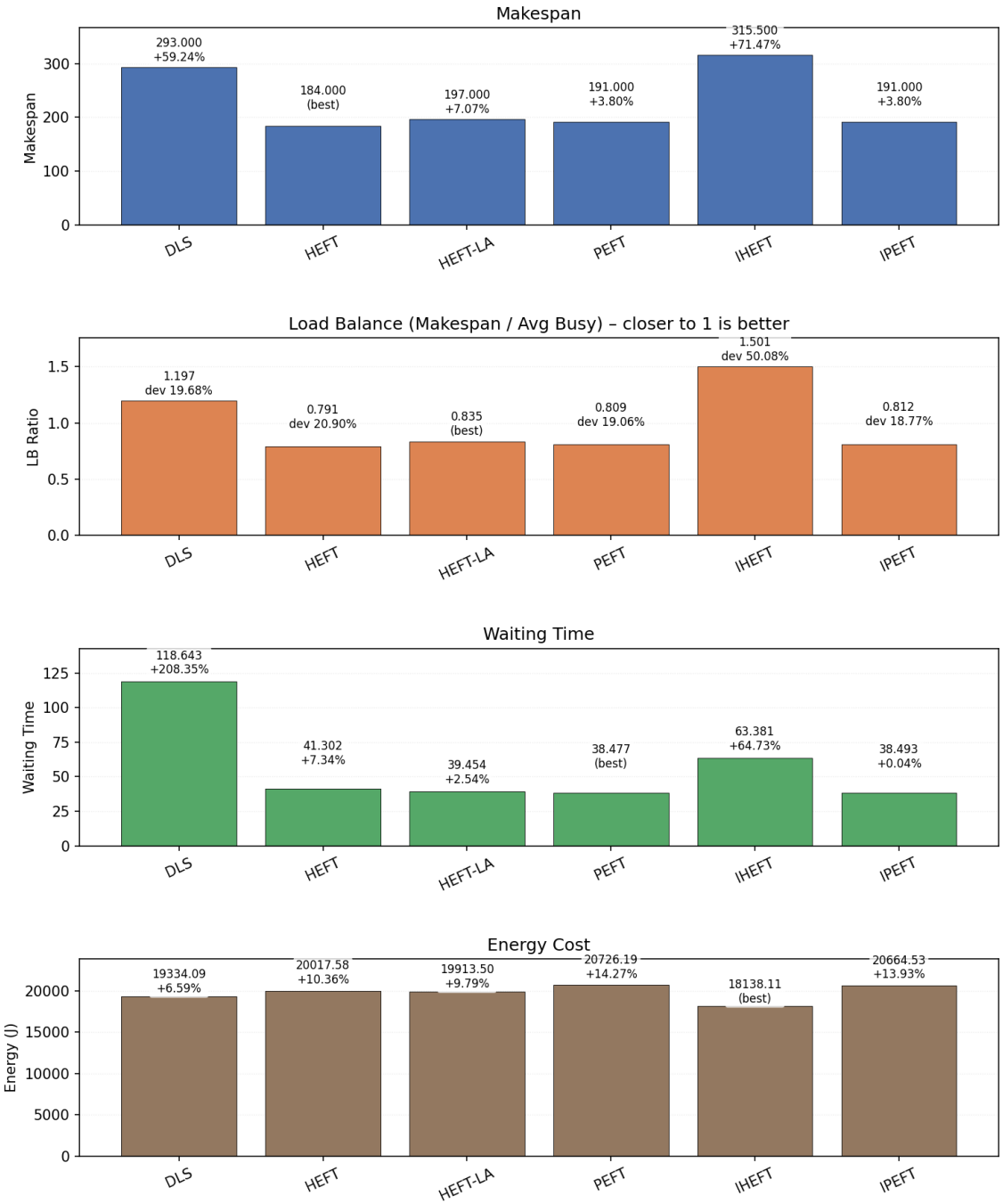


Figura 41 – Métricas (MS, $\bar{U}$, $\overline{\mathrm{WT}}$, E_{total}) – cenário 2, P=32.

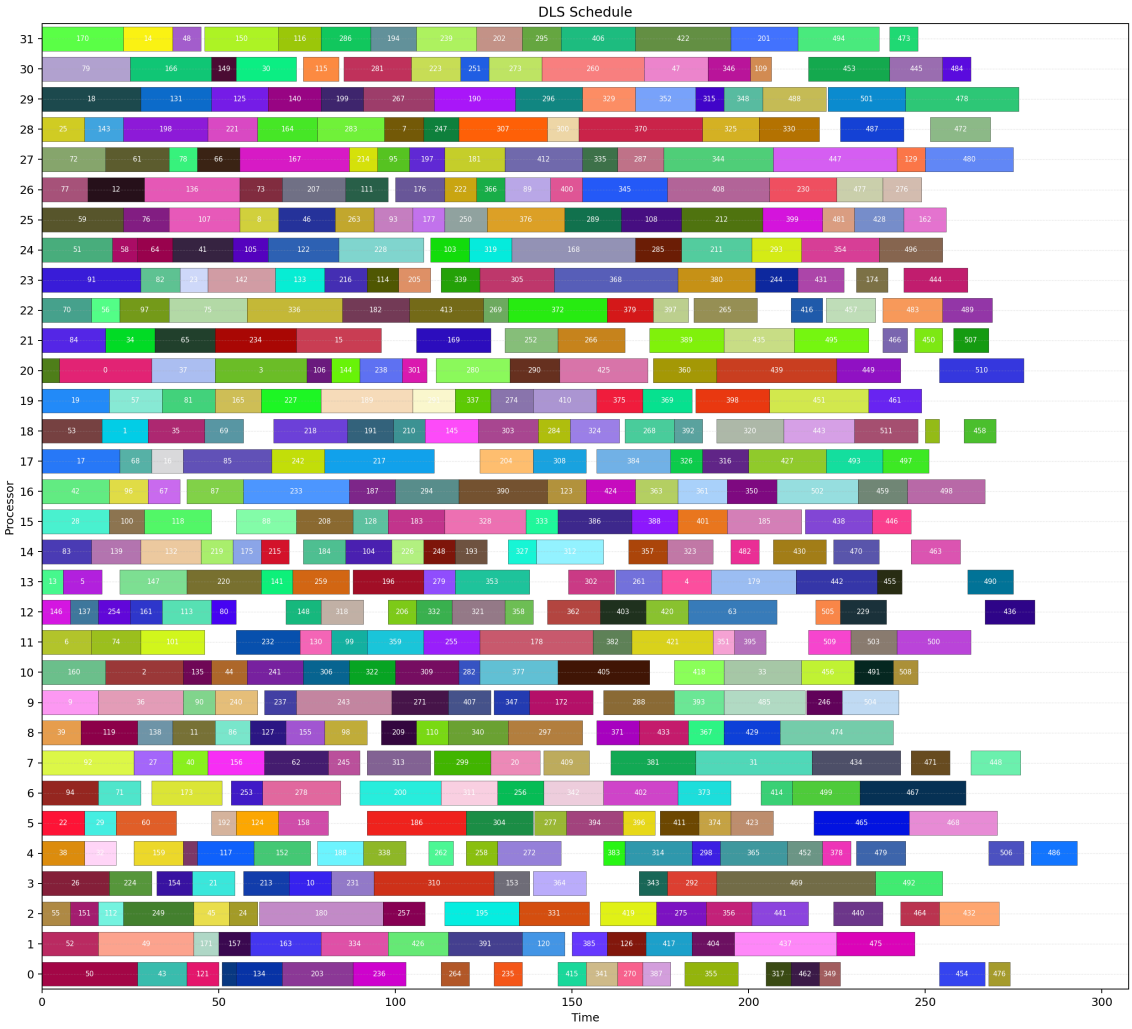


Figura 42 – Gantt – DLS, cenário 2, P=32.

gem), evidenciando boa capacidade de "encher" os *slots* extras sem penalizar o caminho crítico. Já o **IHEFT** permanece sistematicamente atrás em tempo, e o **DLS** apresenta os maiores MS e filas.

Balanceamento de carga ($\bar{U}$). O **DLS** continua exibindo razão $\bar{U}$ próxima de 1 (quase ideal), reforçando sua vocação de equilíbrio global, porém com custo de tempo elevado. Nos demais métodos, $\bar{U}$ é baixo com P=8 — clara concentração das tarefas nos PEs mais rápidos — e melhora com P=16 e 32 (Figura 48b). Um efeito colateral aparece no **IHEFT**: a razão ultrapassa 1 com muitos PEs (sinal de subutilização média frente ao MS), isto é, parte do parque de PEs permanece ociosa enquanto o caminho crítico limita o término.

Energia. Com P=8, o **DLS** tende a consumir menos energia total; a partir de $P \geq 16$, o **IHEFT** passa a ser o mais eficiente energeticamente, consolidando o *trade-off* clássico (economiza energia ao custo de maior MS). Em todos os algoritmos, o aumento de P reduz o consumo total (Figura 48d).



Figura 43 – Gantt – HEFT, cenário 2, P=32.

Leitura dos Gantts. Os cronogramas mostram por que os números acima acontecem: (i) PEFT,IPEFT concentram grandes blocos nos PEs mais rápidos, diminuindo comunicação e filas no caminho crítico; (ii) HEFT,HEFT-LA distribuem melhor quando há mais PEs, aproximando o MS dos líderes sem inflar espera; (iii) **DLS** preenche quase todo o tempo em todos os PEs, mas a fragmentação e as dependências estendem a linha do tempo; (iv) **IHEFT** prioriza mapeamentos energeticamente vantajosos, aceitando janelas ociosas e alongamento do término.

Implicações práticas. Para metas de **tempo de conclusão**, priorize o **PEFT** ou o **IPEFT** com poucos PEs e considere o **HEFT** ou o **HEFT-LA** quando houver mais paralelismo disponível. Para **eficiência energética**, **IHEFT** é a melhor escolha a partir de $P \geq 16$. Se **equilíbrio/fairness** de utilização for requisito (e o tempo puder ser maior), **DLS** entrega o melhor $\bar{U}$.

Relação com o cenário 1. O padrão estrutural se repete: métodos que *concentram* execução nos PEs mais rápidos tendem a otimizar MS e $\overline{\text{WT}}$, enquanto métodos que



Figura 44 – Gantt – HEFT-LA, cenário 2, P=32.

balanceiam globalmente elevam MS; o mapeamento *energy-aware* do **IHEFT** minimiza energia com perda de desempenho. No DAG maior, esses trade-offs ficam ainda mais evidentes, e o ganho de aumentar P satura cedo, pois o caminho crítico passa a dominar o término.

4.4 Cenário 3: $N = 1024$ tarefas

O DAG ilustrativo do cenário 3 está mostrado na Figura 49.

4.4.1 Configuração com $P = 8$

4.4.1.1 Análise P=8

Em Figura 50, **PEFT** e **IPEFT** empatam com o menor MS (≈ 224). Na sequência vêm **HEFT** (≈ 237, +5,8%) e **HEFT-LA** (≈ 244, +8,9%). O **IHEFT** fica bem atrás (≈ 386, +72%), e o **DLS** é muito pior (≈ 2479).



Figura 45 – Gantt – PEFT, cenário 2, P=32.

O **balanceamento** $\bar{U}$ mostra o mesmo *trade-off*: **DLS** é praticamente perfeito ($\bar{U} \approx 1{,}00$), porém com MS altíssimo; as demais heurísticas exibem $\bar{U} \ll 1$ (PEFT/IPEFT $\approx 0{,}09$, HEFT/HEFT-LA $\approx 0{,}10$, IHEFT $\approx 0{,}16$), indicando *concentração de carga* em poucos PEs. Os Gantts confirmam: (i) em Figura 54 e Figura 56, 2–3 processadores dominam o caminho crítico, com blocos longos e pouca ociosidade; (ii) Figura 52 e Figura 53 distribuem melhor, mas preservam nós críticos nos PEs mais rápidos; (iii) Figura 51 exibe alta fragmentação e "buracos" de precedência/comunicação, alongando a linha do tempo total; (iv) já Figura 55 evidencia janelas ociosas prolongadas em PEs menos velozes, coerentes com o MS elevado da heurística.

No **tempo médio de espera** $\overline{\mathrm{WT}}$, **IPEFT** é o melhor ($\approx$ 48,4) e **PEFT** fica colado ($\approx$ 48,6); **HEFT** e **HEFT-LA** mantêm filas moderadas ($\approx$ 50–52); **IHEFT** aumenta a espera ($\approx$ 76) e **DLS** é muito elevado ($\approx$ 1275).

Quanto à **energia** E_{total}, as diferenças são pequenas: **IHEFT** atinge o menor valor ($\approx 4{,}93 \times 10^4$); {PEFT, IPEFT, HEFT} ficam cerca de +2,5% acima do mínimo e



Figura 46 – Gantt – IHEFT, cenário 2, P=32.

HEFT-LA é o mais custoso ($\approx$ +4,8%). Assim, *IHEFT* favorece levemente a eficiência energética, mas com penalidade temporal expressiva.

4.4.1.2 Síntese P=8

PEFT e **IPEFT** entregam o menor MS ($\approx$ 224) e a menor $\overline{\text{WT}}$, com leve aumento de energia frente ao mínimo. **HEFT** e **HEFT-LA** ficam próximos em tempo, mas não superam as variantes com inserção. **IHEFT** minimiza E_{total} com tempo maior. **DLS** é dominado em métricas de tempo, apesar do ótimo $\bar{U}$. Em heterogeneidade acentuada, priorizar os PEs mais rápidos e encaixar tarefas críticas via inserção é mais efetivo do que balancear globalmente.



Figura 47 – Gantt – IPEFT, cenário 2, P=32.

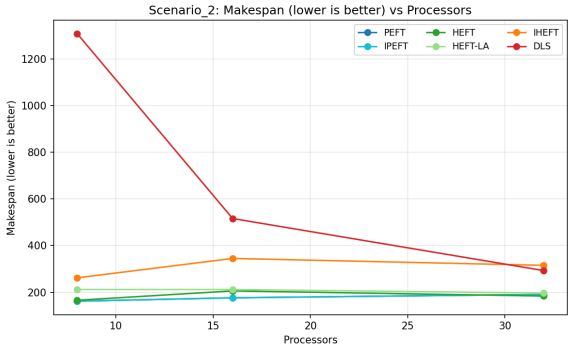
4.4.2 Configuração com $P = 16$

4.4.2.1 Análise P=16

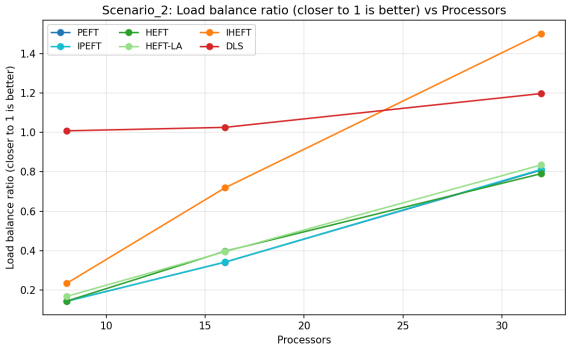
A partir de Figura 57, os algoritmos baseados em *insertion* e precedência (**PEFT** e **IPEFT**) seguem entregando os *menores makespans*, com **HEFT-LA** e **HEFT** logo atrás. O **IHEFT** permanece sensivelmente pior em tempo total e o **DLS** é o mais lento.

O **balanceamento de carga** (razão makespan/busy médio; quanto mais próximo de 1, melhor o balanceamento) reforça o padrão visto nos Gantts:

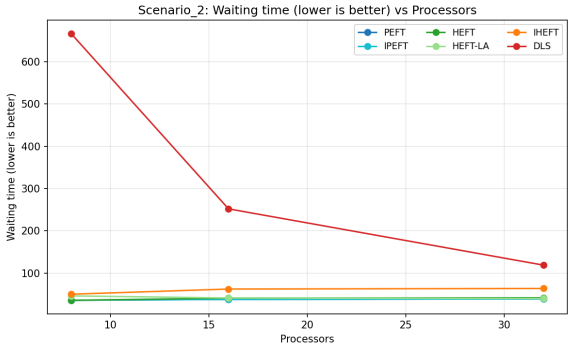
- No Figura 58, quase não há ociosidade – o balanceamento é ótimo –, mas a fragmentação intensa e a comunicação/precedências ao longo do caminho crítico alongam a linha do tempo.
- Nos Gantts de **PEFT** e **IPEFT** (Figura 61 e Figura 63), há *concentração de tarefas* em um subconjunto dos PEs mais rápidos; vários PEs ficam menos utilizados após



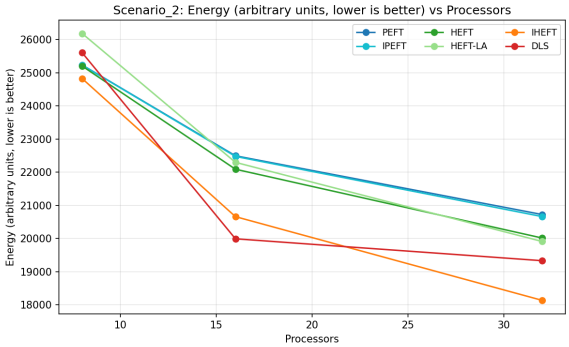
(a) Makespan por número de processadores (↓ melhor).



(b) Razão de balanceamento de carga (≈ 1 é melhor).



(c) Tempo médio de espera $\overline{\mathrm{WT}}$ (↓ melhor).



(d) Energia total E_{total} (↓ melhor).

Figura 48 – Síntese visual do cenário 2: evolução das métricas com $P \in \{8, 16, 32\}$.

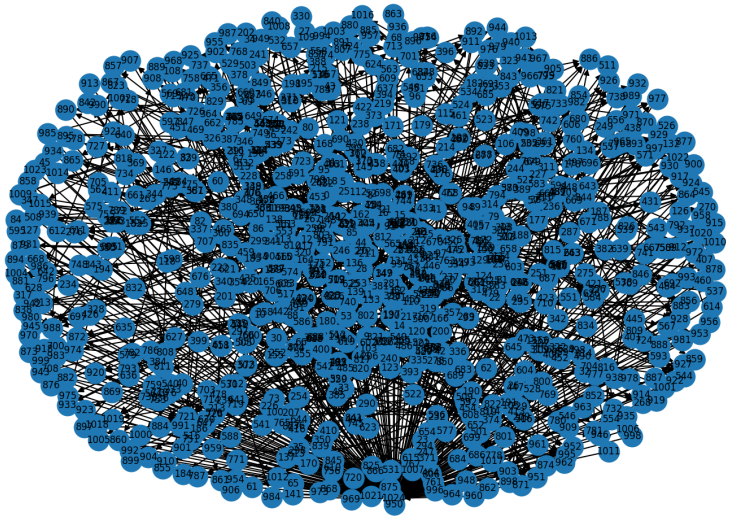


Figura 49 – DAG ilustrativo do cenário 3 (N=1024).

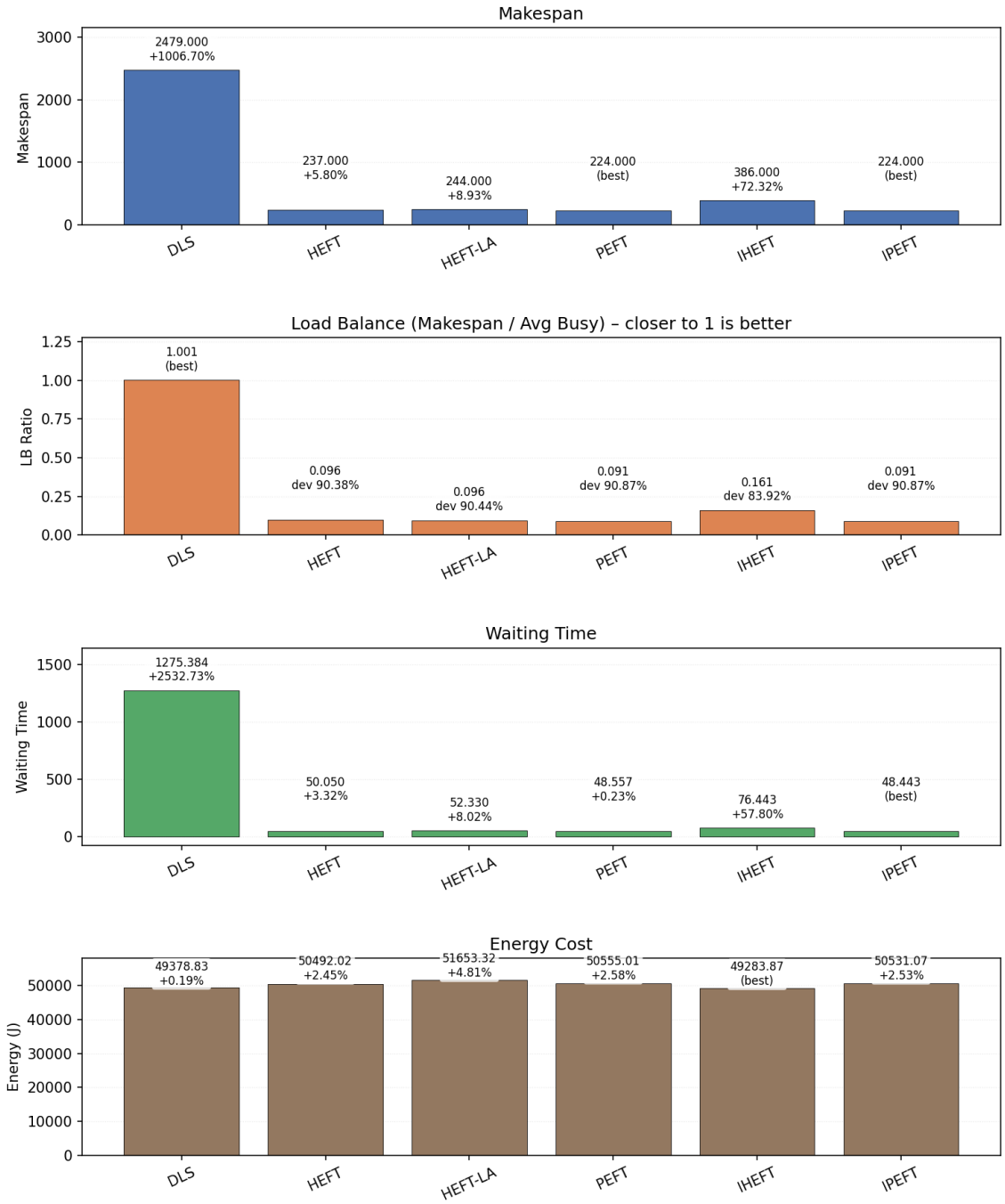


Figura 50 – Métricas (MS, $\bar{U}$, $\overline{\mathrm{WT}}$, E_{total}) — cenário 3, P=8.

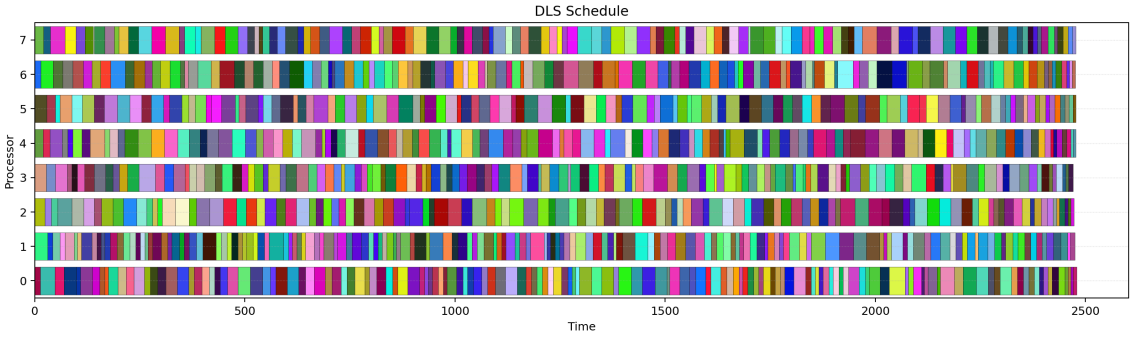


Figura 51 – Gantt — DLS, cenário 3, P=8.

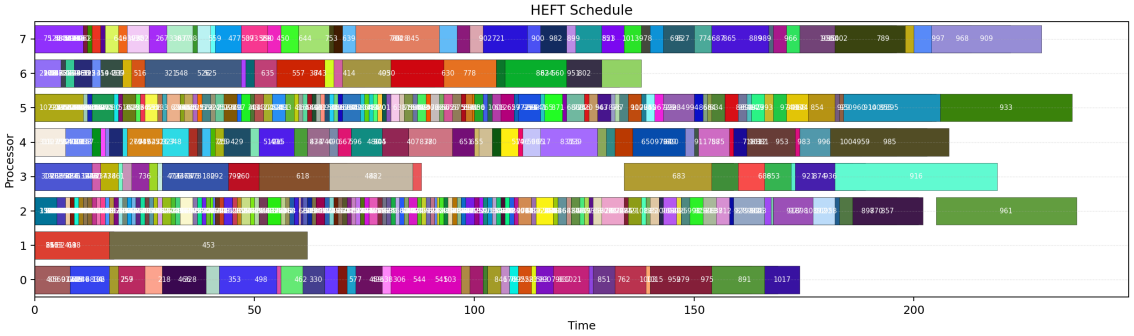


Figura 52 – Gantt — HEFT, cenário 3, P=8.

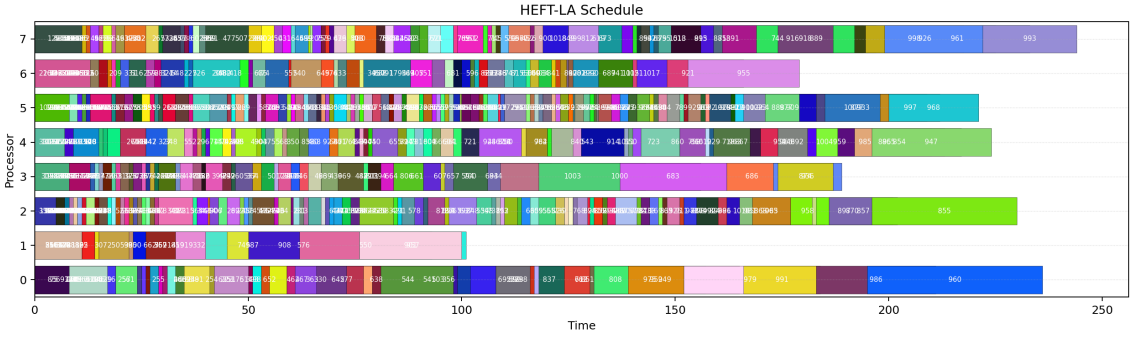


Figura 53 – Gantt — HEFT-LA, cenário 3, P=8.

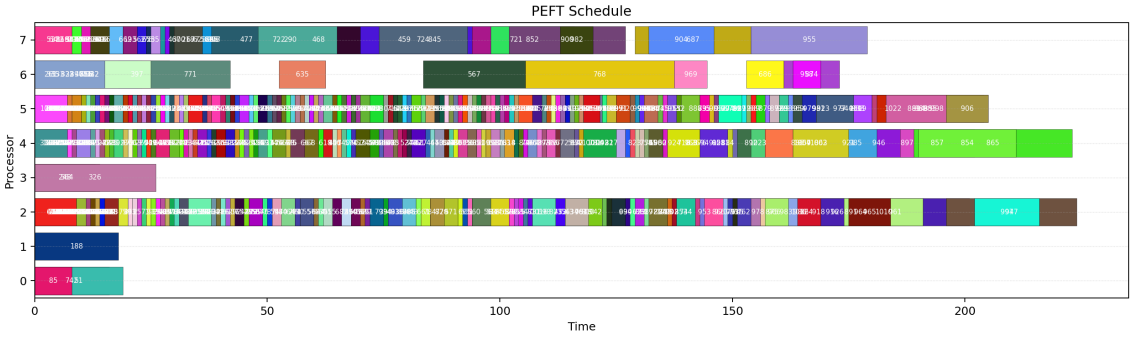


Figura 54 – Gantt — PEFT, cenário 3, P=8.

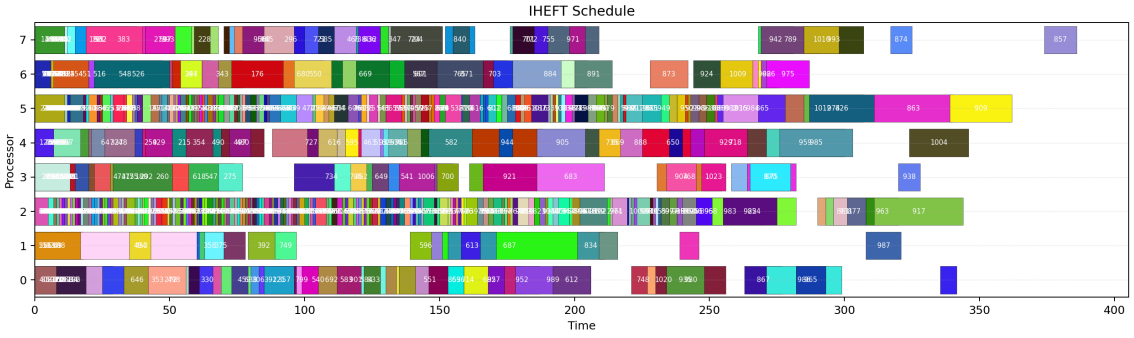


Figura 55 – Gantt — IHEFT, cenário 3, P=8.

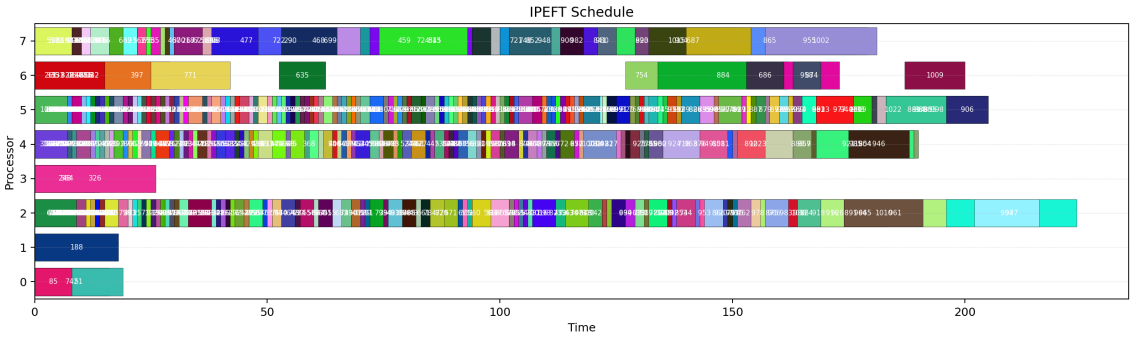


Figura 56 – Gantt — IPEFT, cenário 3, P=8.

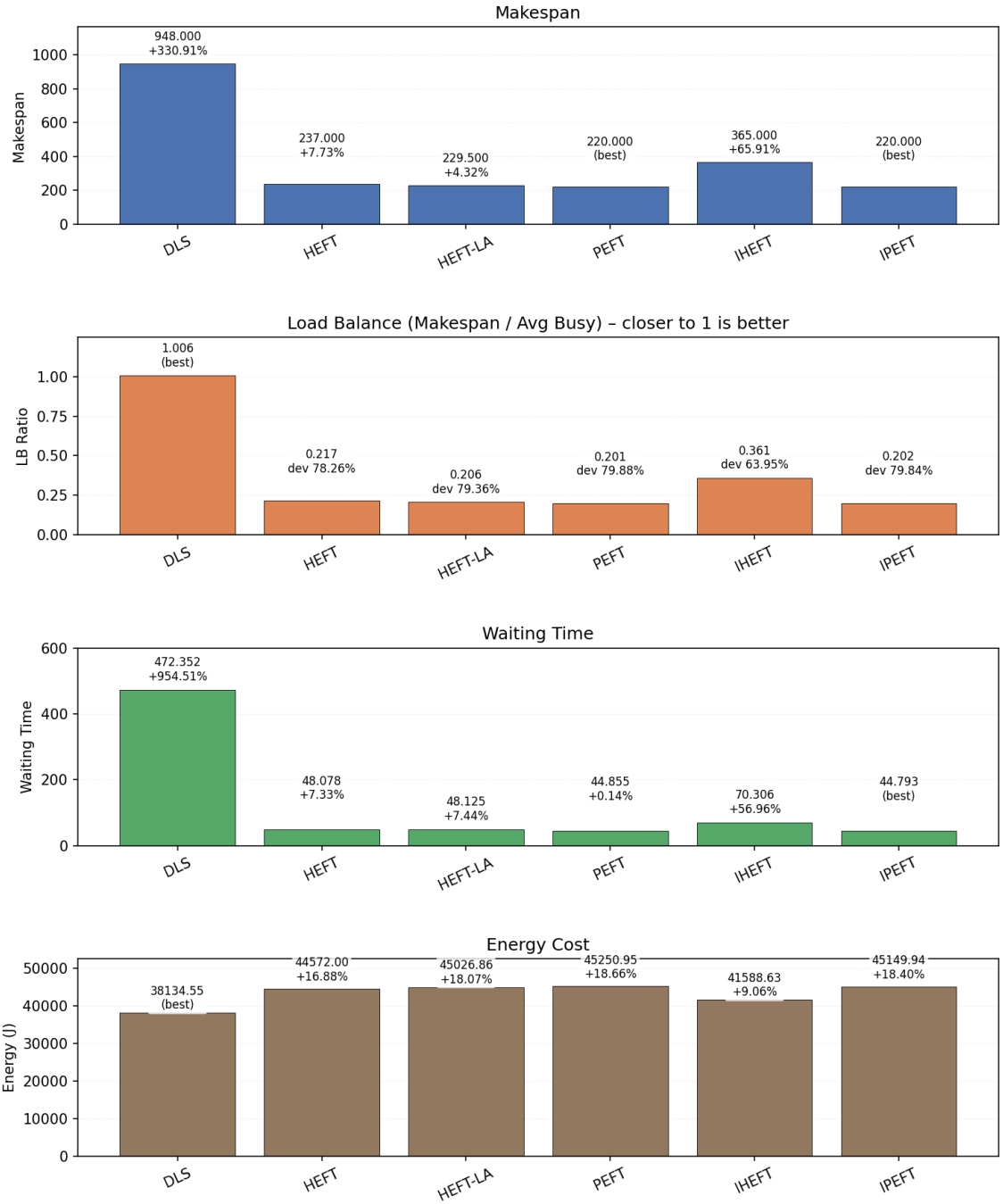


Figura 57 – Métricas (MS, $\bar{U}$, $\overline{\mathrm{WT}}$, E_{total}) — cenário 3, P=16.

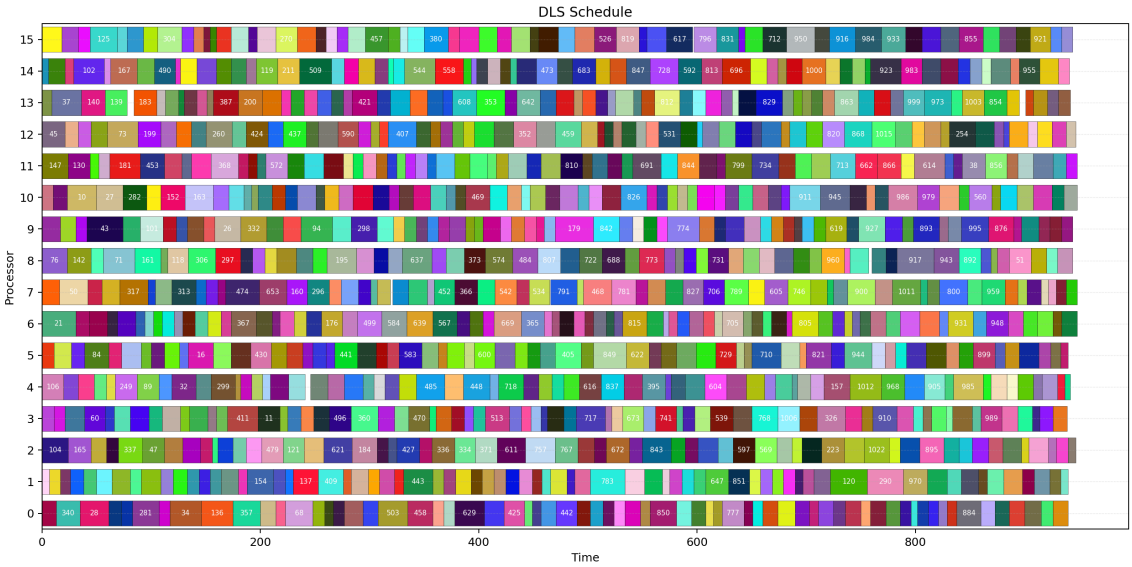


Figura 58 – Gantt – DLS, cenário 3, P=16.

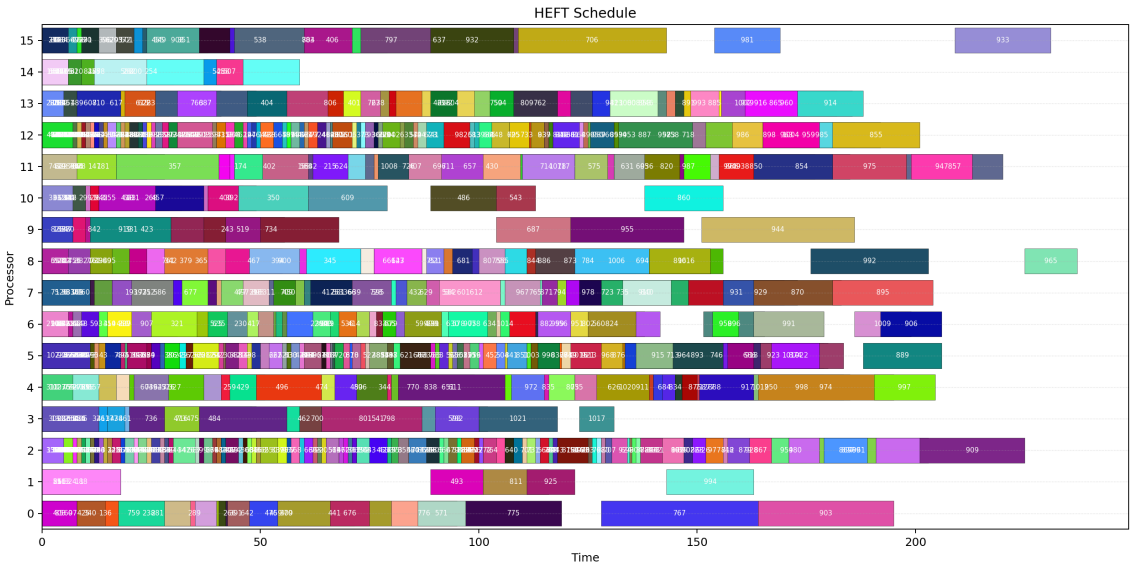


Figura 59 – Gantt – HEFT, cenário 3, P=16.

a metade da execução. O balanceamento global aparenta pior, porém o *makespan* é menor por explorar melhor a heterogeneidade.

- **HEFT** e **HEFT-LA** (Figura 59 e Figura 60) espalham mais o trabalho, mas ainda com blocos longos em PEs preferenciais, ficando entre os extremos de DLS e PEFT/IPEFT.

- Já o Figura 62 evidencia tarefas tardias isoladas e janelas ociosas, coerentes com $\overline{\text{WT}}$ e MS mais altos.

No **tempo médio de espera**, **IPEFT** e **PEFT** tendem a apresentar as menores filas graças às políticas de inserção; **HEFT** e **HEFT-LA** ficam próximos; **IHEFT** eleva

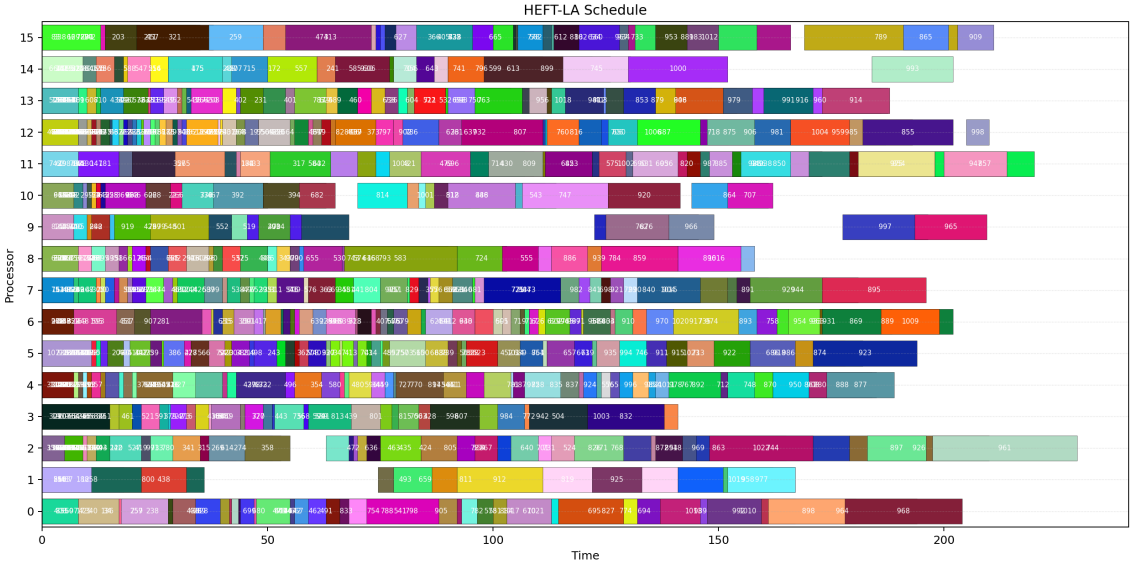


Figura 60 – Gantt – HEFT-LA, cenário 3, P=16.

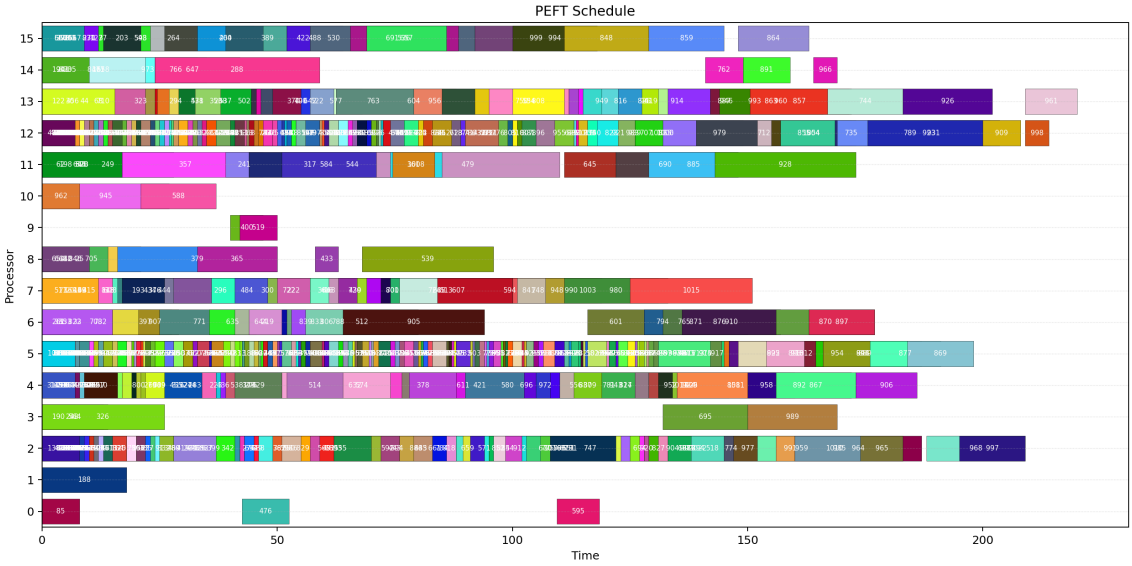


Figura 61 – Gantt – PEFT, cenário 3, P=16.

as esperas e **DLS** volta a ser o pior.

Quanto ao **custo de energia total**, as diferenças entre os quatro melhores em tempo (**PEFT**, **IPEFT**, **HEFT-LA**, **HEFT**) permanecem relativamente pequenas. Em geral, **IHEFT** e/ou **DLS** ficam ligeiramente melhor em energia, mas à custa de um tempo total significativamente maior.

4.4.2.2 Síntese P=16

PEFT e **IPEFT** seguem como as escolhas preferenciais pelo menor makespan e menores tempos de espera; **HEFT-LA** e **HEFT** são alternativas próximas, com execução mais espalhada e estabilidade. **IHEFT** e principalmente **DLS** ficam atrás em tempo total;

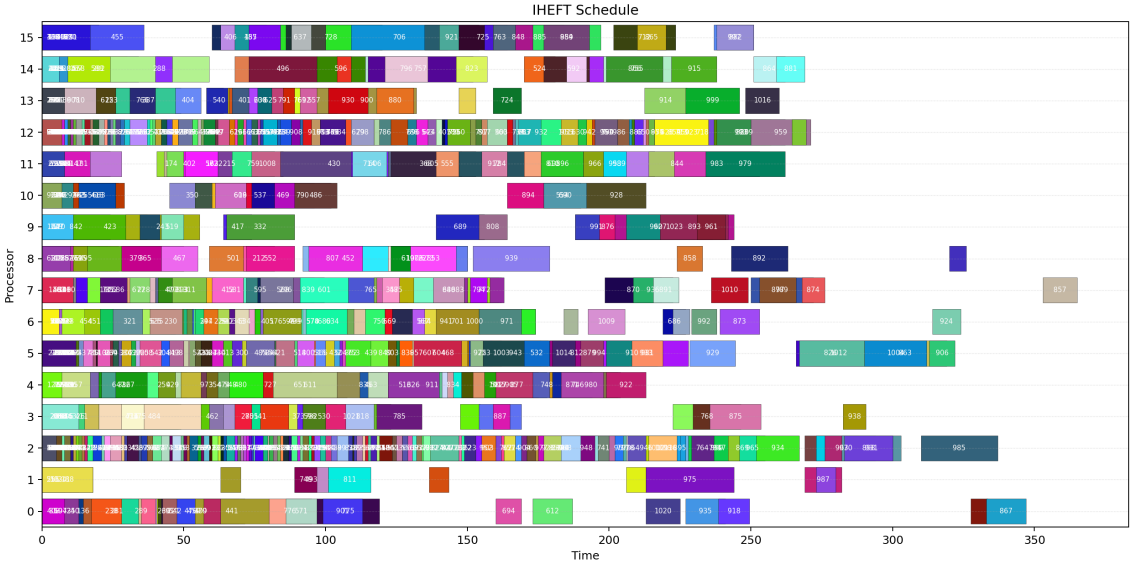


Figura 62 – Gantt – IHEFT, cenário 3, P=16.

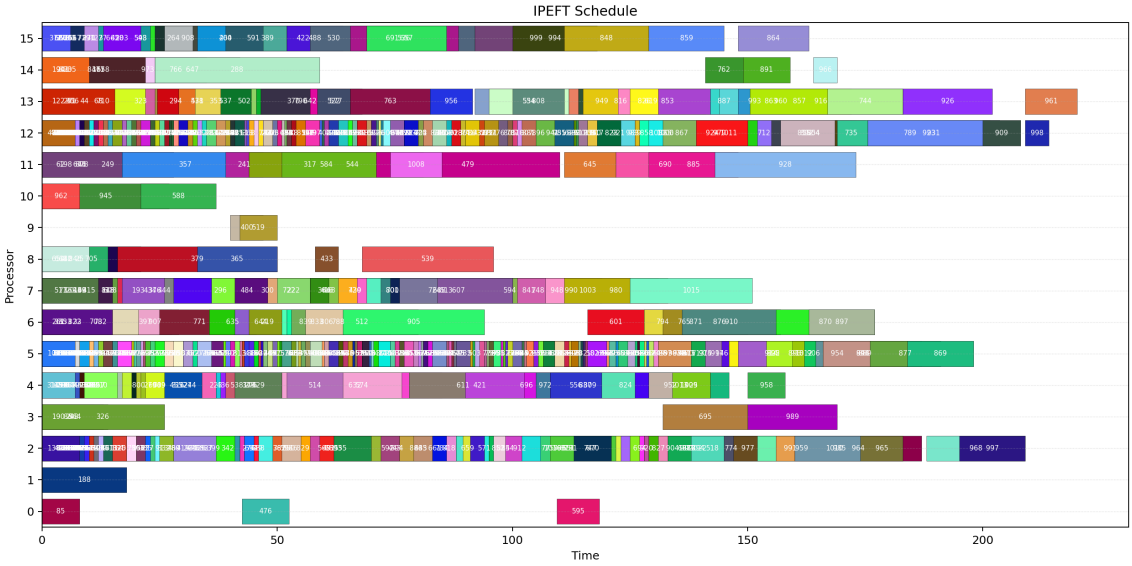


Figura 63 – Gantt – IPEFT, cenário 3, P=16.

o eventual leve ganho energético não compensa o aumento de duração.

4.4.3 Configuração com $P = 32$

4.4.3.1 Análise P=32

Em Figura 64, o **IPEFT** obtém o menor MS (≈ 210), seguido de **PEFT** (≈ 219, +4,3%), **HEFT-LA** (≈ 220, +4,8%) e **HEFT** ($\approx 222{,}5$, +6,0%). **IHEFT** cresce bem mais (≈ 371, +76,7%) e o **DLS** é o pior (≈ 438, +108,6%).

O **balanceamento médio** $\bar{U}$ evidencia o já visto nos demais valores de P deste cenário: **DLS** fica praticamente perfeito ($\bar{U} \approx 1{,}03$), mas não traduz isso em menor MS;

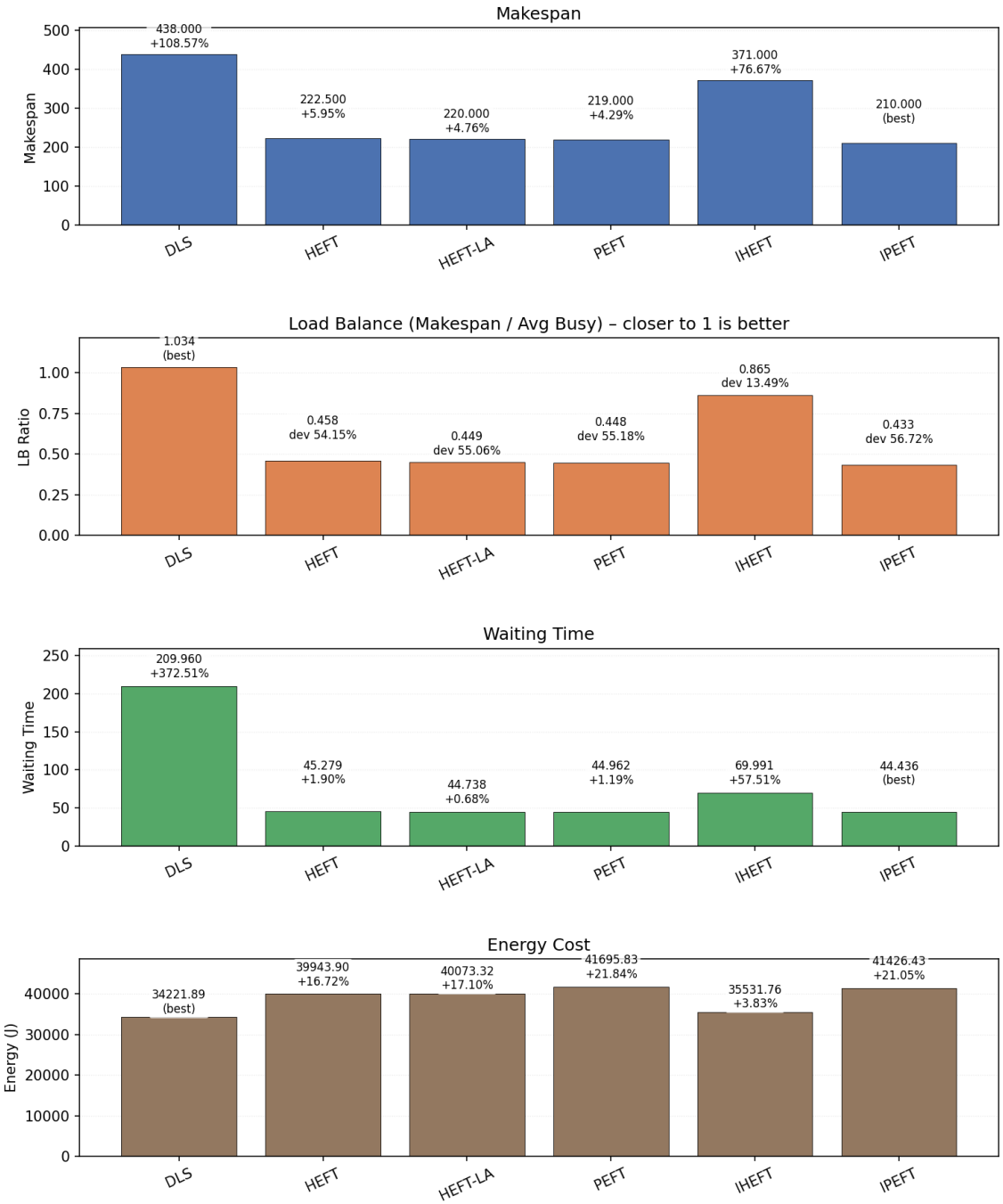


Figura 64 – Métricas (MS, $\bar{U}$, $\overline{\mathrm{WT}}$, E_{total}) – cenário 3, P=32.

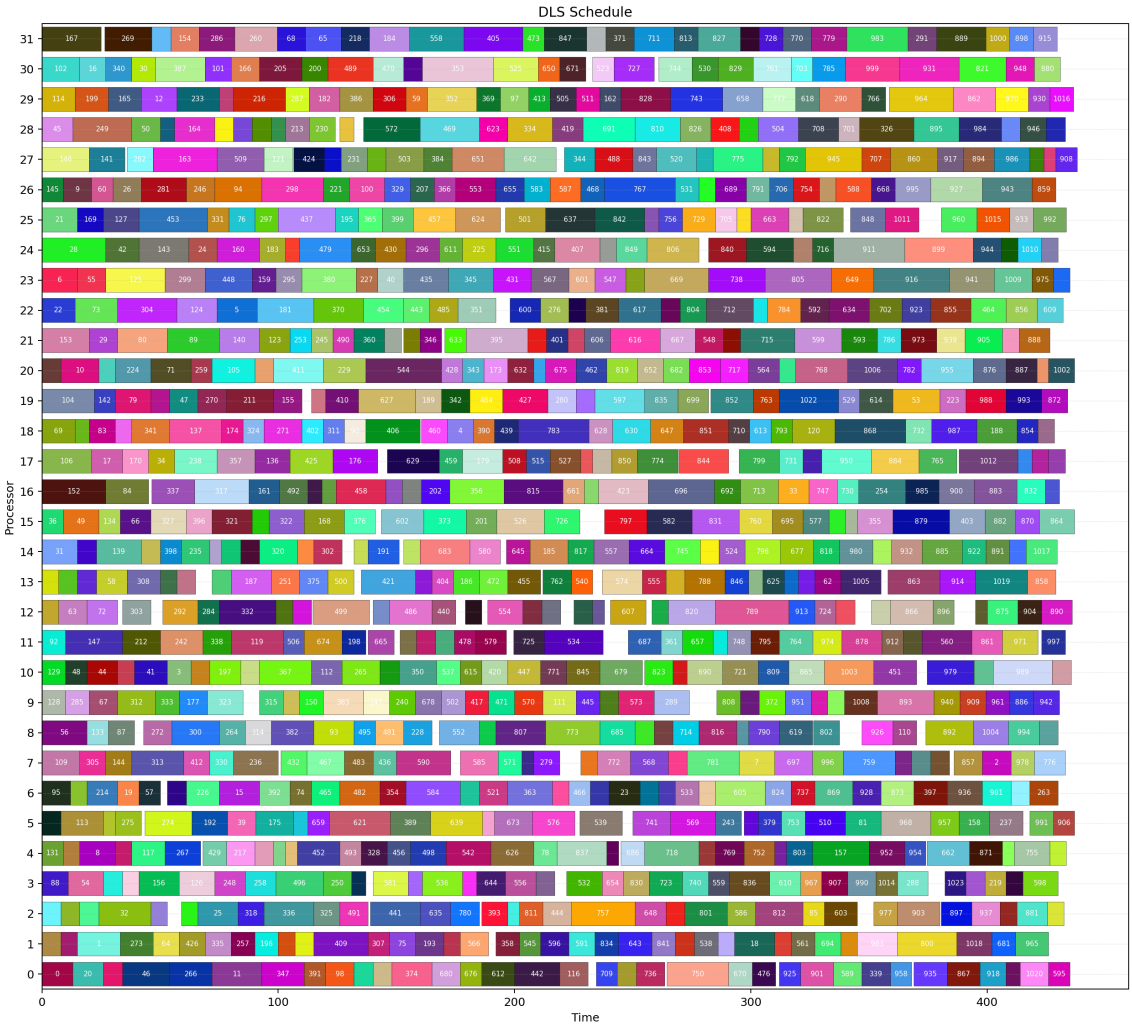


Figura 65 – Gantt – DLS, cenário 3, P=32.

IHEFT é o segundo mais "equilibrado" ($\approx 0{,}87$), enquanto {**PEFT**, **IPEFT**, **HEFT**, **HEFT-LA**} operam com $\bar{U} \in [0{,}43, 0{,}46]$, isto é, forte concentração do trabalho em poucos PEs. Os Gantts confirmam: no Figura 65 a ocupação é quase contínua e fragmentada em *muitas* tarefas curtas, ao passo que Figura 68 e Figura 70 exibem *faixas longas* em alguns PEs rápidos e ociosidade precoce nos demais — estratégia que encurta o caminho crítico; padrão semelhante e competitivo aparece em Figura 66 e Figura 67, enquanto Figura 69 mostra blocos em PEs mais lentos e cauda longa.

No **tempo médio de espera** $\overline{\mathrm{WT}}$, **IPEFT** é o melhor ($\approx 44{,}4$), com **HEFT-LA** ($\approx 44{,}7$, $+0{,}7\%$) e **PEFT** ($\approx 45{,}0$, $+1{,}2\%$) praticamente empatados; **HEFT** vem logo depois ($\approx 45{,}3$, $+1{,}9\%$). **IHEFT** piora consideravelmente ($\approx 70{,}0$), e o **DLS** salta para ≈ 210.

Quanto ao **custo energético** E_{total}, o **DLS** é o menor ($3{,}42\times10^4$), seguido de **IHEFT** ($3{,}55\times10^4$, $+3{,}8\%$). As variantes focadas em tempo ficam $+16\%$–22% acima do mínimo.



Figura 66 – Gantt – HEFT, cenário 3, P=32.

4.4.3.2 Síntese P=32

IPEFT entrega o menor MS e a menor $\overline{\text{WT}}$, com **PEFT** e **HEFT-LA** e **HEFT** muito próximos. **IHEFT** e, sobretudo, **DLS** ficam atrás em tempo, embora **DLS** minimize E_{total} e apresente o $\bar{U}$ mais próximo de 1.

4.4.4 Síntese do cenário 3 (N=1024)

Os gráficos de Figura 71a mostram que **IPEFT**, **PEFT** e **HEFT-LA** formam o pelotão de frente em makespan para todo P, com diferenças pequenas entre si; **HEFT** fica logo atrás. **IHEFT** apresenta tempos maiores e o **DLS** é substancialmente pior, embora reduza bastante ao aumentar P. Entre 16 e 32 processadores, os ganhos de {IPEFT, PEFT, HEFT-LA} tendem a saturar, sinalizando que o caminho crítico domina a execução e que mais paralelismo agregado traz retorno marginal.

A razão de balanceamento (Figura 71b) evidencia a natureza do mapeamento: **DLS** opera muito próximo de 1, distribuindo a carga de forma quase uniforme; **IHEFT**

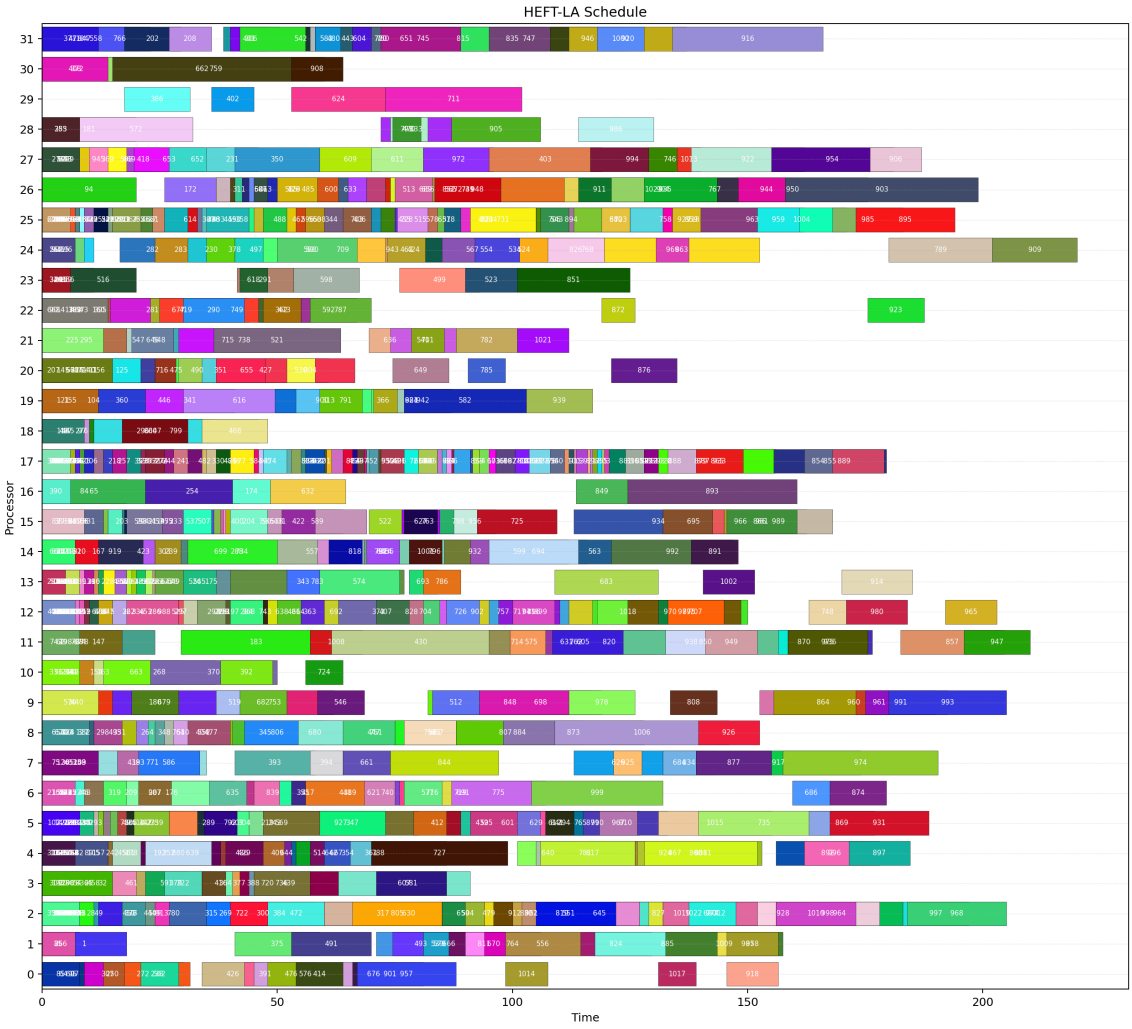


Figura 67 – Gantt – HEFT-LA, cenário 3, P=32.

melhora rapidamente com P; já { **IPEFT**, **PEFT**, **HEFT**, **HEFT-LA** } mantêm $\bar{U}$ bem abaixo de 1, mesmo com mais processadores — as heurísticas que entregam os menores makespans *concentram* trabalho nos PEs mais rápidos, em vez de balancear globalmente.

No **tempo médio de espera** (Figura 71c), **IPEFT** é tipicamente o melhor, seguido de **PEFT** e **HEFT-LA**, depois **HEFT**; **IHEFT** mantém filas maiores e **DLS** apresenta esperas muito elevadas com P pequeno, caindo à medida que P cresce — reforçando o efeito das políticas de inserção/antecipação.

Quanto ao **custo energético** (Figura 71d), **DLS** e **IHEFT** tendem a consumir menos energia total à medida que P cresce, enquanto { **HEFT**, **HEFT-LA**, **PEFT**, **IPEFT** } permanecem em patamar mais alto — coerente com a exploração agressiva dos PEs mais rápidos para reduzir MS.

Em resumo: para *tempo-de-treinamento*, {**IPEFT**, **PEFT**, **HEFT-LA**} são as escolhas naturais, com ganhos que se estabilizam de 16 para 32 processadores por limitação de caminho crítico/overheads. Para *custo energético*, {**DLS**, **IHEFT**} aparecem



Figura 68 – Gantt – PEFT, cenário 3, P=32.

vantajosos, aceitando makespans bem maiores. O padrão estratégico — melhores tempos com baixa $\bar{U}$ via concentração em poucos PEs rápidos versus melhor energia com maior $\bar{U}$ — replica os demais cenários (sem comparação direta de valores) e reforça a importância de alinhar a heurística ao objetivo final.

4.5 Cenário 4: $N = 2048$ tarefas

4.5.0.1 Configuração com $P = 8$

4.5.0.2 Análise P=8

Em Figura 73, **PEFT** e **IPEFT** voltam a liderar o MS (empate em $\approx$ 222) e também reduzem a $\overline{\text{WT}}$ ($\approx$ 40). Duas novidades deste cenário com N=2048: (i) o **DLS** apresenta $\bar{U} \approx 1$ mas mantém MS *muito* alto (padrão de cauda longa – Figura 74), evidenciando que $\bar{U}$ próximo de 1 não garante tempo total baixo quando o caminho crítico é mal explorado; (ii) em **energia**, o **DLS** é o menor E_{total} (diferença de apenas 0–6% para

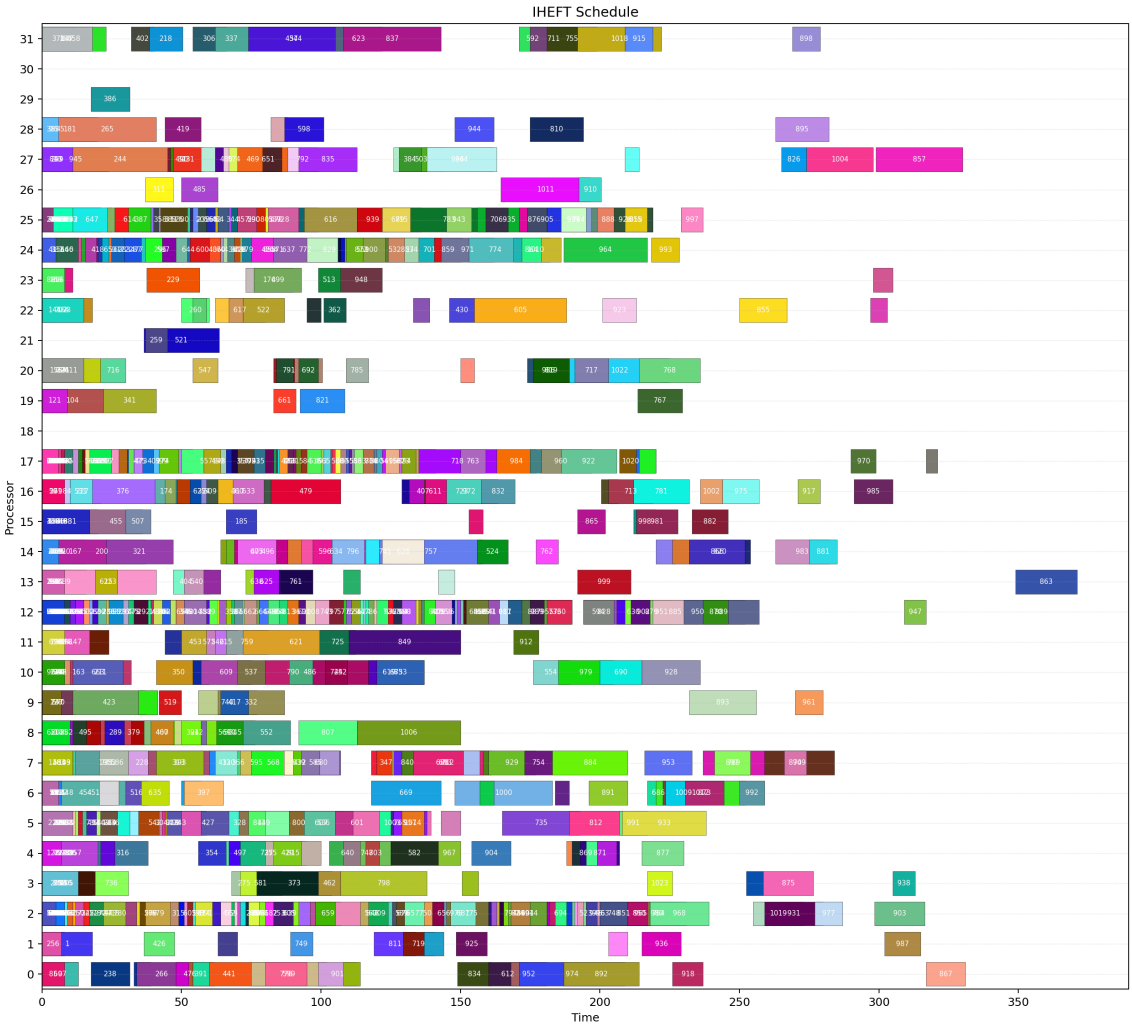


Figura 69 – Gantt – IHEFT, cenário 3, P=32.

os demais), enquanto **IHEFT** fica próximo do mínimo porém com penalidade temporal. Entre as heurísticas de lista, **HEFT-LA** não traz ganho adicional neste P=8 (caudas ligeiramente maiores), e **IHEFT** reduz "buracos" mas introduz terminações tardias, elevando MS e $\overline{\text{WT}}$. Em suma, com muita tarefa e poucos PEs, priorizar o caminho crítico (PEFT/IPEFT) supera balanceamento global; eficiência energética do DLS/IHEFT não compensa o aumento expressivo no tempo.

4.5.0.3 Configuração com $P = 16$

4.5.0.4 Análise P=16

Em Figura 80, **IPEFT** obtém o menor MS ($\approx$ 232), seguido de **PEFT** ($\approx$ 235). **HEFT** e **HEFT-LA** empatam ($\approx$ 252), sem ganho adicional do look-ahead. **IHEFT** fica acima ($\approx$ 417) e **DLS** mantém MS elevado ($\approx$ 1836).

No **balanceamento** $\bar{U}$, apenas **DLS** fica perto de 1; as demais heurísticas permanecem com $\bar{U}$ baixo (PEFT/IPEFT $\approx$ 0.115, HEFT/HEFT-LA $\approx$ 0.12, IHEFT $\approx$ 0.22),

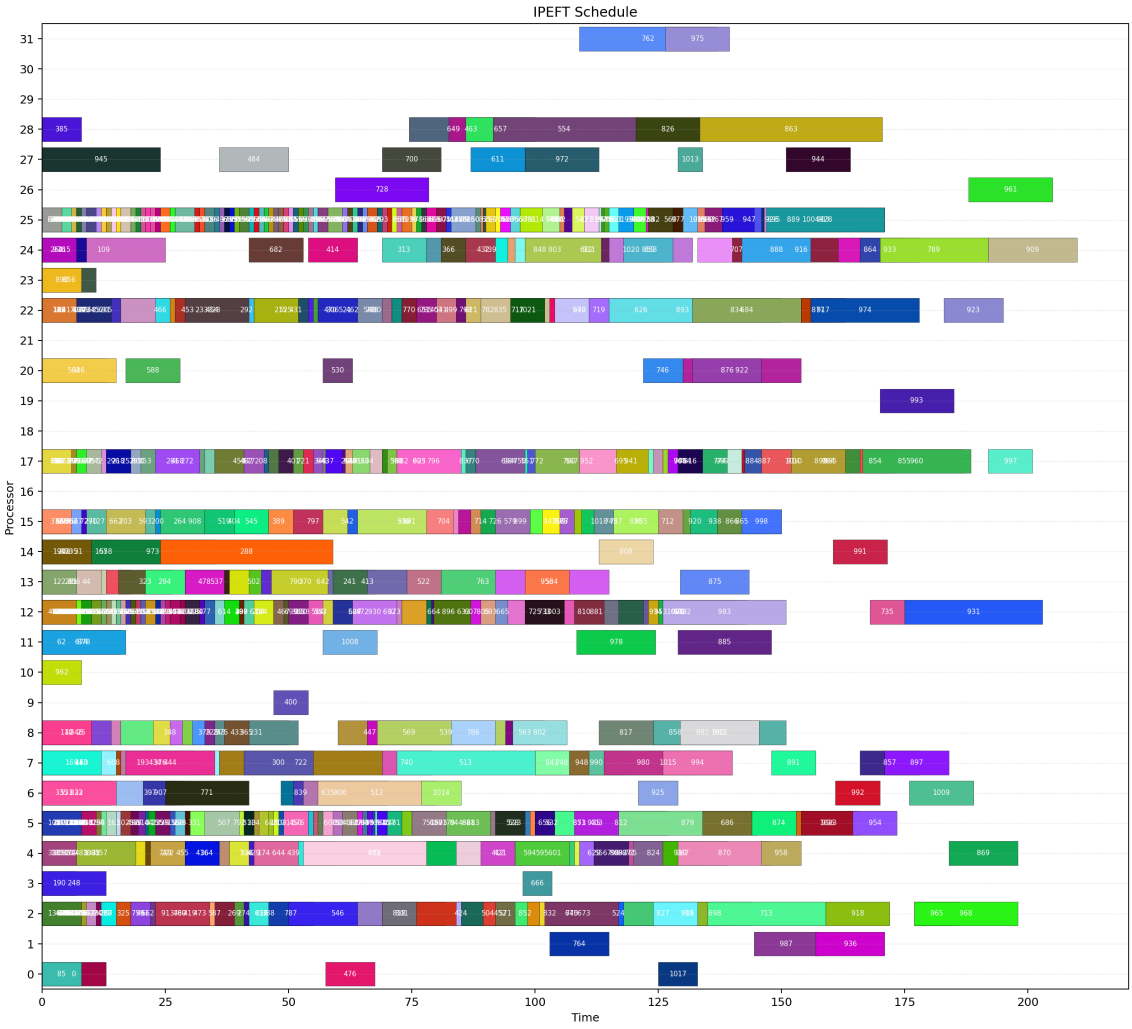


Figura 70 – Gantt – IPEFT, cenário 3, P=32.

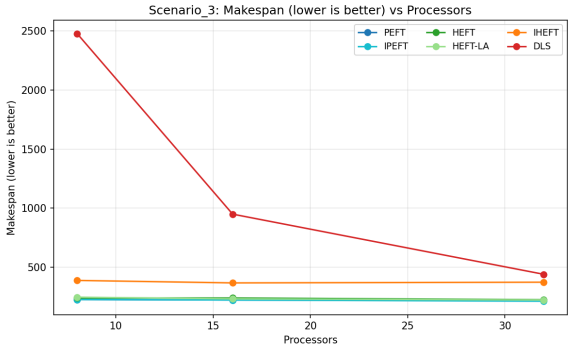
indicando que encurtar o caminho crítico é mais determinante que equilibrar globalmente. Os Gantts (Figura 84–Figura 86) mostram PEFT/IPEFT concentrando tarefas críticas em poucos PEs, HEFT/HEFT-LA mais espalhados, e Figura 81 com forte fragmentação.

Na **espera média** $\overline{\mathrm{WT}}$, **PEFT** e **IPEFT** são os menores (≈ 46), HEFT/HEFT-LA vêm logo atrás ($\approx$ 49–52) e **IHEFT** apresenta filas maiores (≈ 77). Em **energia** E_{total}, **DLS** tem o menor consumo (ordem de 7.46×10^4); as heurísticas com inserção e HEFT ficam cerca de +18% a +20% acima desse mínimo, e **IHEFT** fica intermediário ($\approx +10\%$). Em resumo: com P=16, **IPEFT** lidera em tempo, **PEFT** é muito próximo, HEFT/HEFT-LA empatam ligeiramente acima, enquanto **DLS** preserva vantagem energética apesar do MS alto.

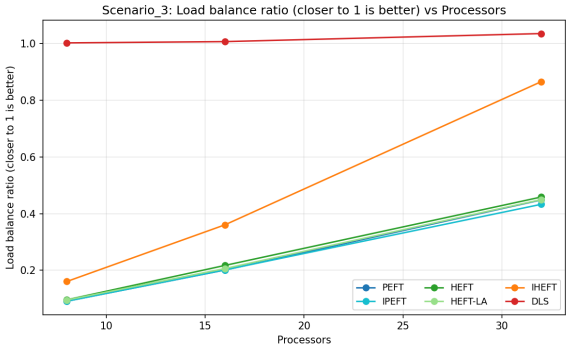
4.5.0.5 Configuração com $P = 32$

4.5.0.6 Análise P=32

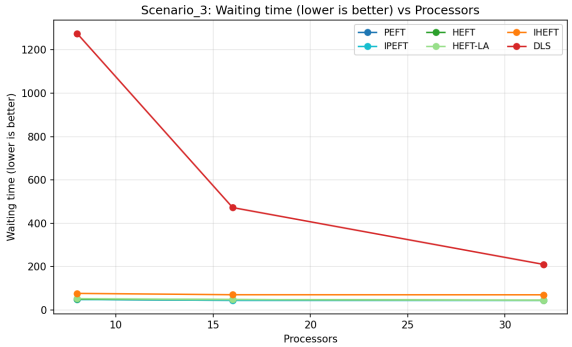
Os principais pontos, complementando as análises anteriores, são:



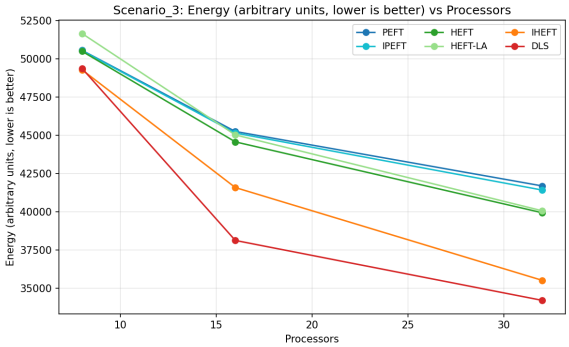
(a) Makespan por número de processadores (↓ melhor).



(b) Razão de balanceamento de carga (≈ 1 é melhor).



(c) Tempo médio de espera $\overline{\mathrm{WT}}$ (↓ melhor).



(d) Energia total E_{total} (↓ melhor).

Figura 71 – Síntese visual do cenário 3: evolução das métricas com $P \in \{8, 16, 32\}$.

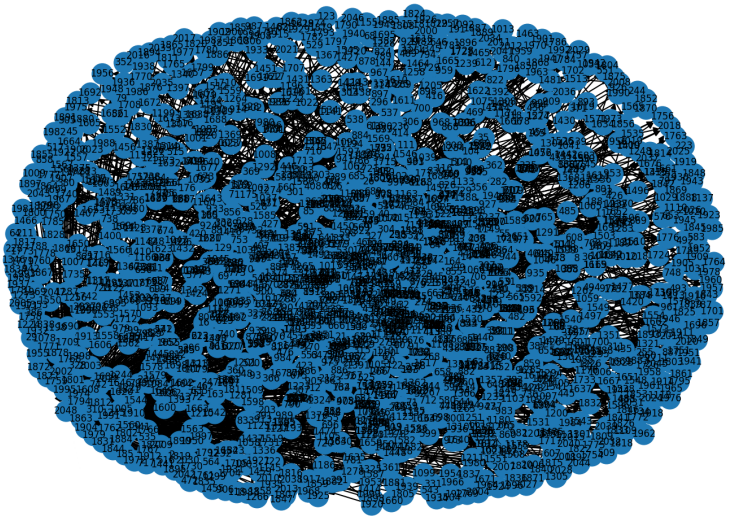


Figura 72 – DAG ilustrativo do cenário 4 (N=2048).

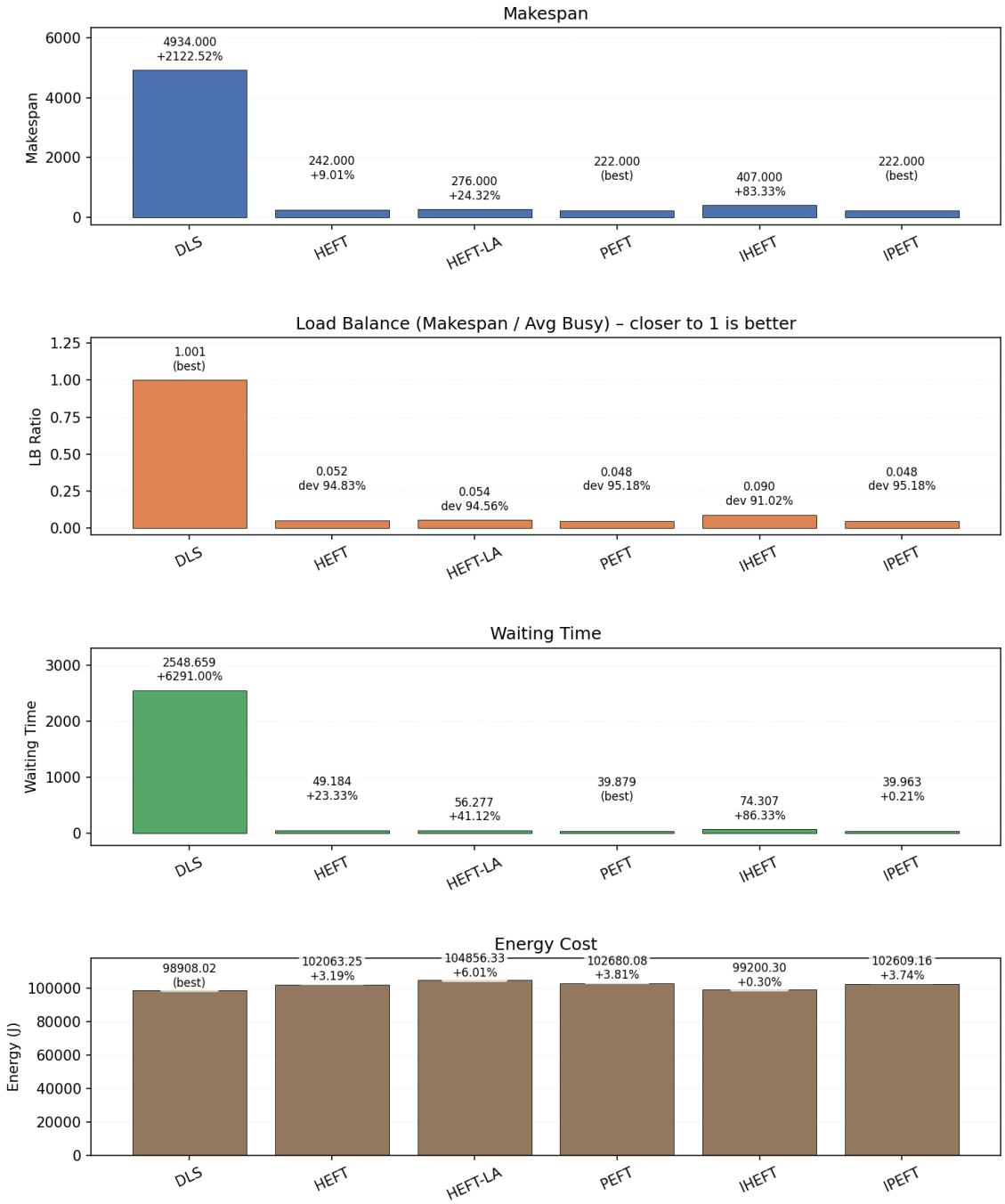


Figura 73 – Métricas (MS, $\bar{U}$, $\overline{\mathrm{WT}}$, E_{total}) – cenário 4, P=8.

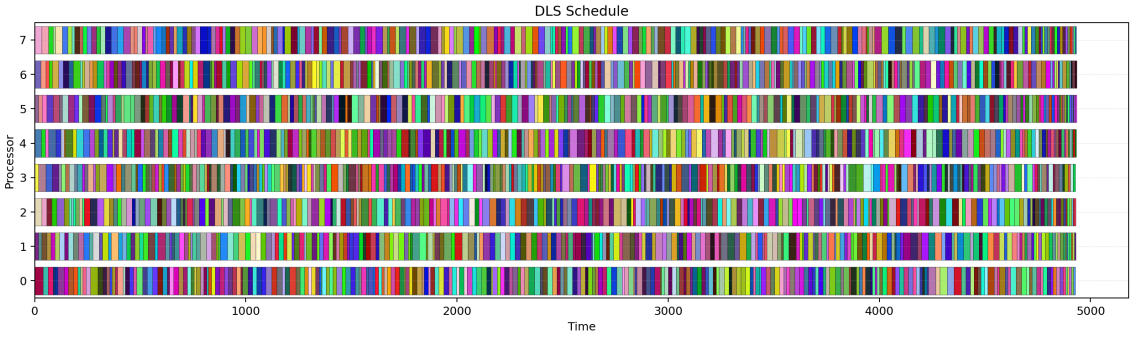


Figura 74 – Gantt – DLS, cenário 4, P=8.

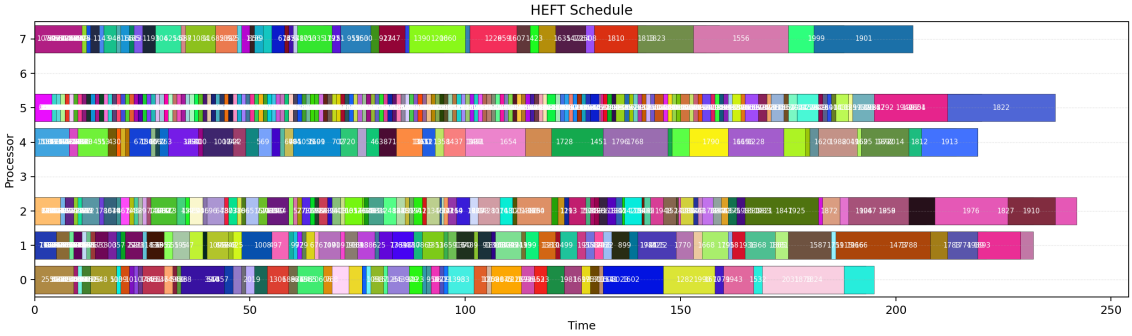


Figura 75 – Gantt – HEFT, cenário 4, P=8.

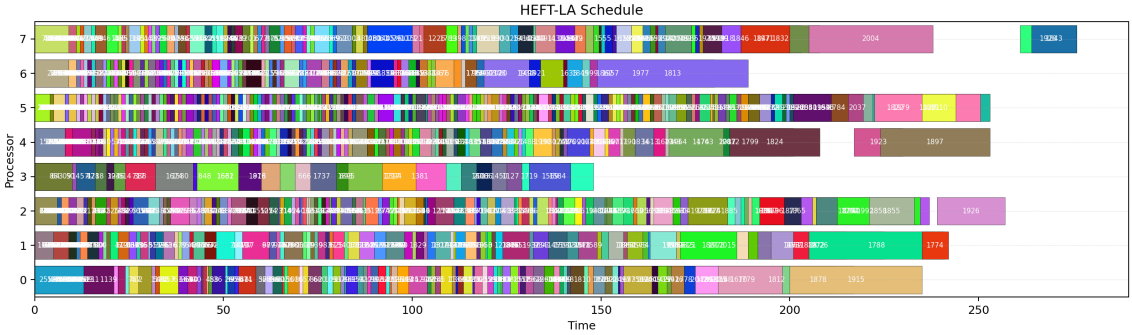


Figura 76 – Gantt – HEFT-LA, cenário 4, P=8.

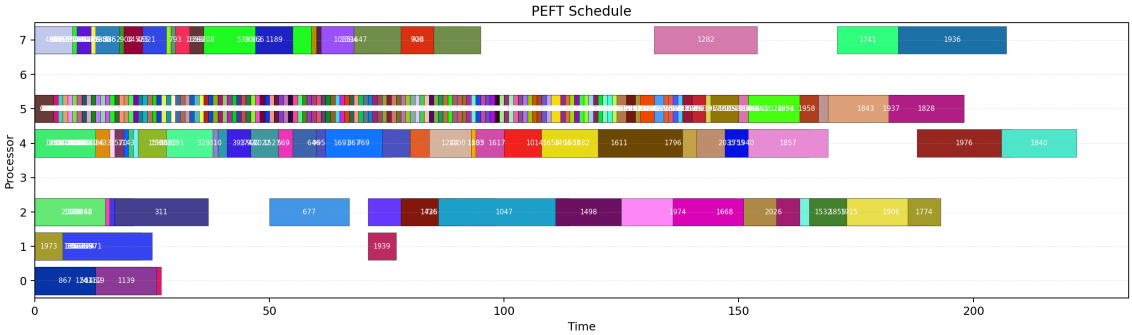


Figura 77 – Gantt – PEFT, cenário 4, P=8.

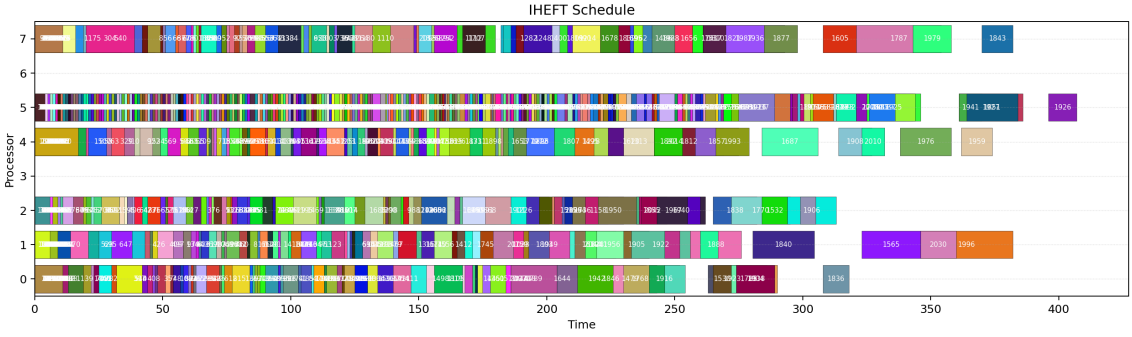


Figura 78 – Gantt – IHEFT, cenário 4, P=8.

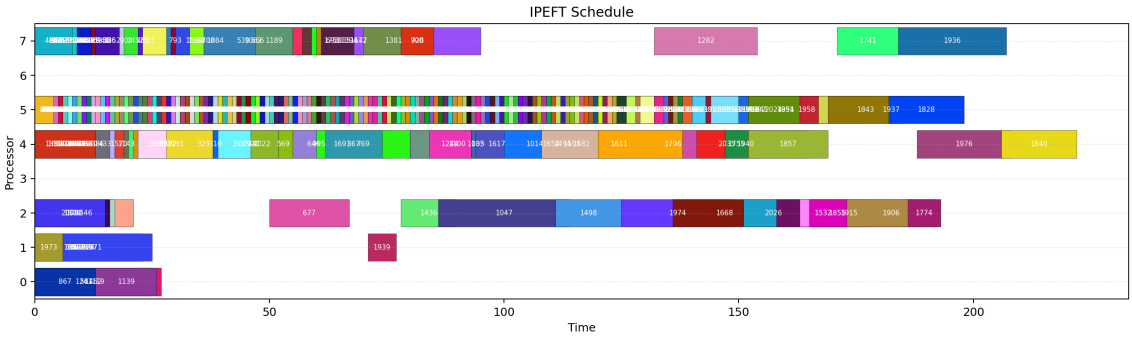


Figura 79 – Gantt – IPEFT, cenário 4, P=8.

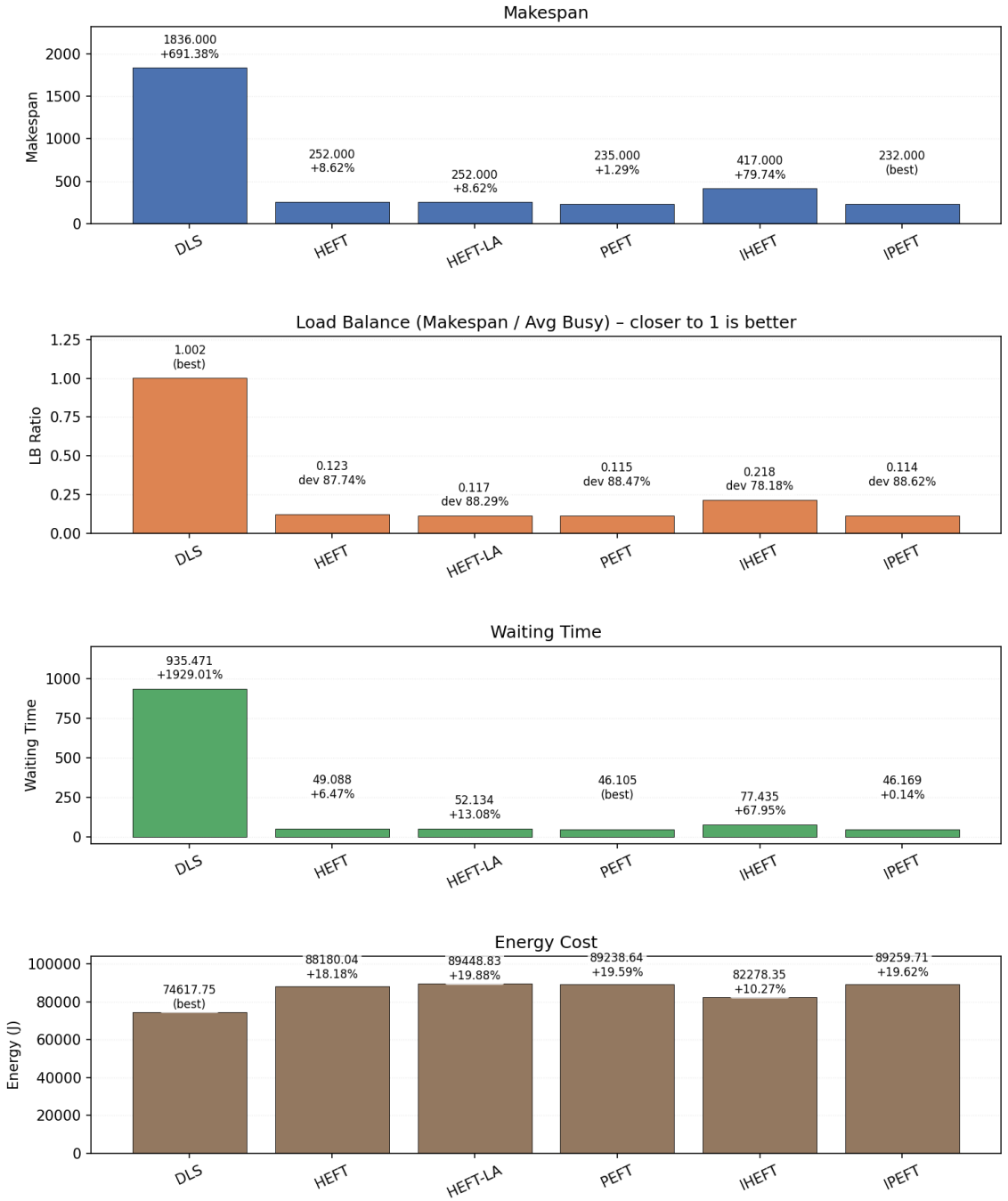


Figura 80 – Métricas (MS, $\bar{U}$, $\overline{\mathrm{WT}}$, E_{total}) — cenário 4, P=16.

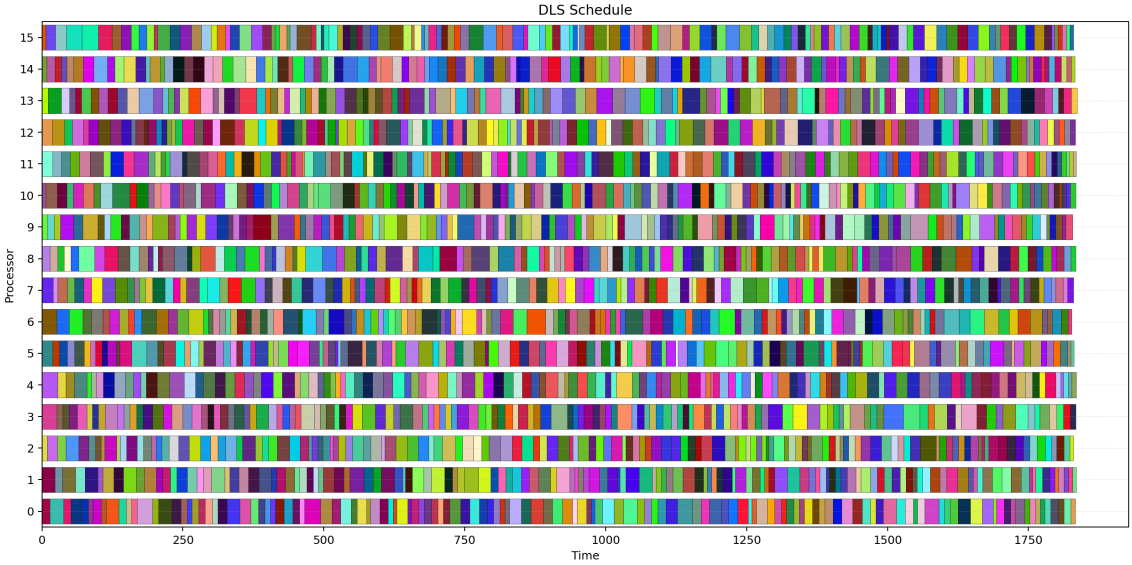


Figura 81 – Gantt — DLS, cenário 4, P=16.

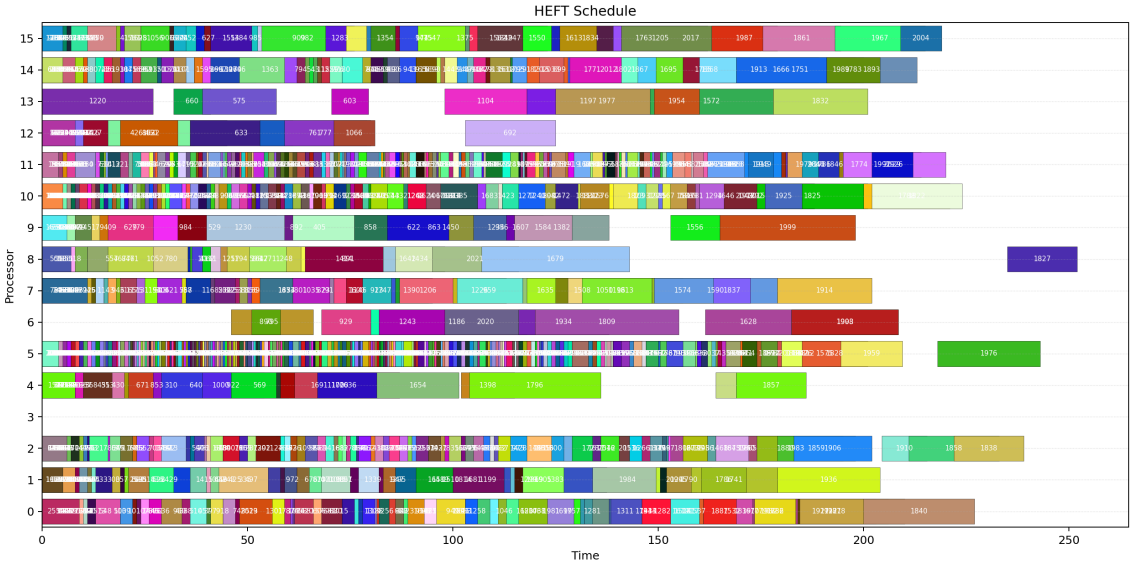


Figura 82 – Gantt — HEFT, cenário 4, P=16.

- **Tempo**: com 32 PEs, as heurísticas baseadas em *lookahead* mantêm o melhor MS. O **HEFT-LA** aparece ligeiramente à frente, com **HEFT** e **IPEFT** e **PEFT** muito próximos. A diferença entre esses quatro métodos torna-se pequena, sugerindo *saturação por precedência/comunicação* em vez de falta de paralelismo.

- **Balanceamento**: o **DLS** segue com $\bar{U} \approx 1$, porém sem converter em menor MS (caminho crítico permanece nos PEs mais rápidos). Para as demais heurísticas, $\bar{U}$ melhora com P mas fica distante de 1, coerente com os Gantts: poucos PEs concentram as tarefas críticas (Figura 89–Figura 91).

- **Espera**: $\overline{\mathrm{WT}}$ estabiliza em torno de valores modestos para HEFT/HEFT-LA/PEFT/IPEFT, indicando filas curtas mesmo com P elevado; o **DLS** reduz a espera em relação a

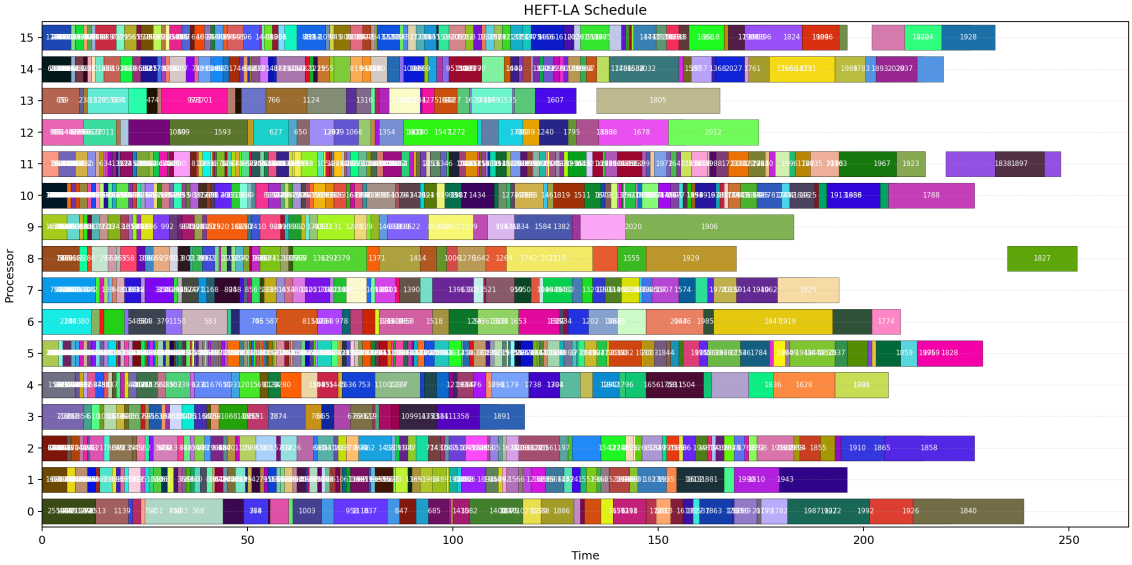


Figura 83 – Gantt — HEFT-LA, cenário 4, P=16.

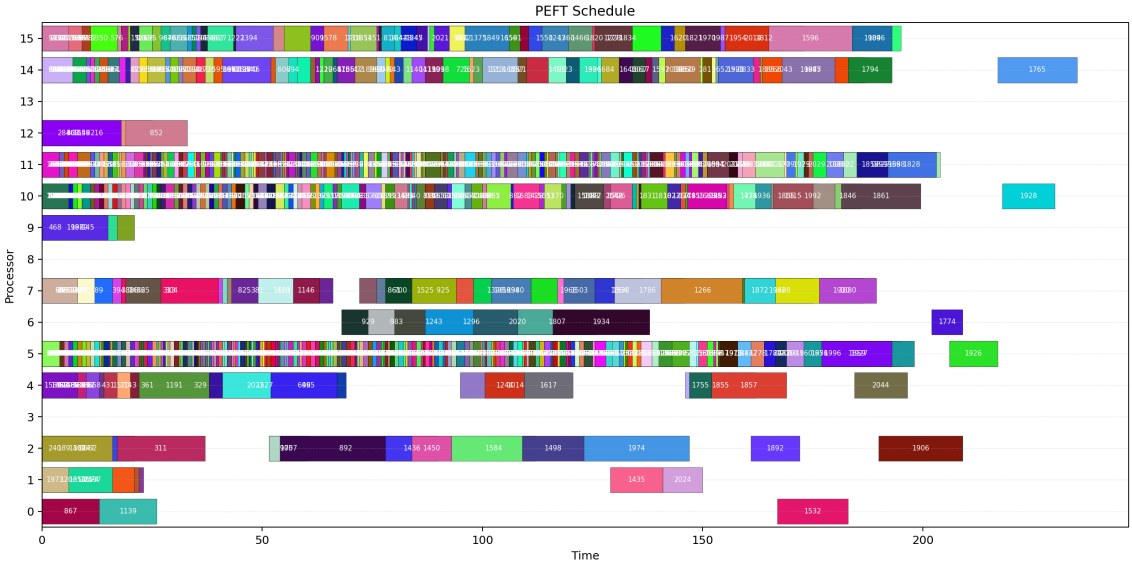


Figura 84 – Gantt — PEFT, cenário 4, P=16.

P=8 e P=16, mas ainda fica muito acima dos demais.

- **Energia**: a ordenação observada antes se mantém – **DLS** tipicamente com o menor E_{total}, seguido por **IHEFT**; {HEFT, HEFT-LA} e {PEFT, IPEFT} aparecem próximos entre si e acima do mínimo. Ou seja, *ganhos de tempo via inserção/lookahead não vêm acompanhados de ganhos de energia.*

- **Efeito novo com** 32 **PEs**: surgem *janelas de ociosidade* nos Gantts mesmo em heurísticas rápidas, causadas por dependências longas e custos de comunicação; ampliar P além desse ponto traz retornos marginais no MS, reforçando que o *gargalo é de precedência*, não de capacidade.



Figura 85 – Gantt — IHEFT, cenário 4, P=16.

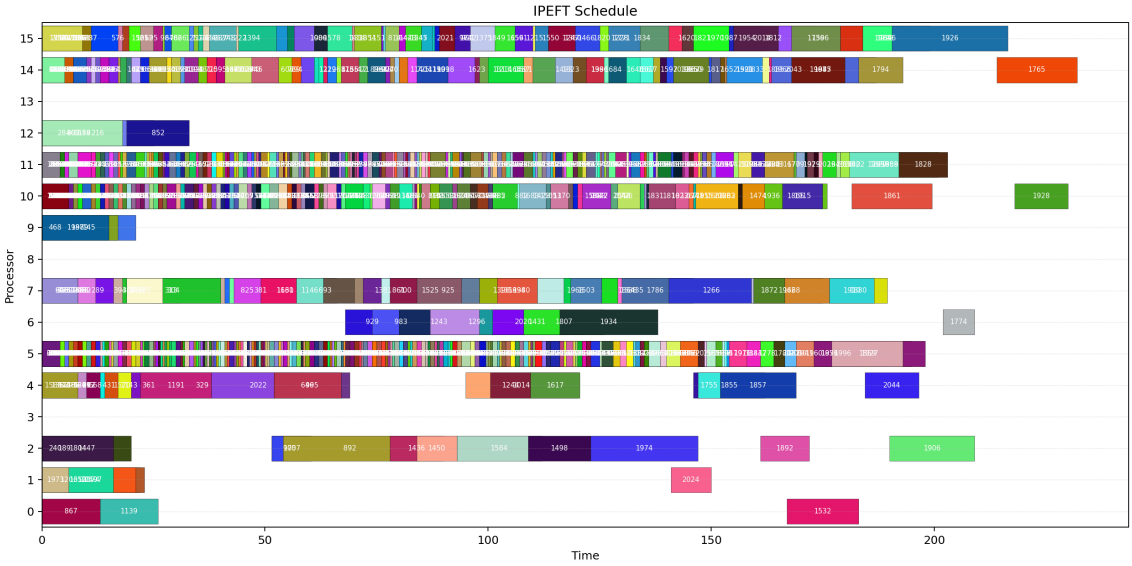


Figura 86 – Gantt — IPEFT, cenário 4, P=16.

4.5.1 Síntese do cenário 4

4.5.1.0.1 Principais achados.

(i) **Tempo** (Figura 94a): as heurísticas com *inserção* (**PEFT** e **IPEFT**) permanecem entre as melhores em MS para todos os valores de P, com **HEFT-LA** muito próxima (e por vezes empatando). **IHEFT** melhora ao aumentar P, mas segue com MS superior ao grupo anterior. **DLS** cai bastante com mais processadores, porém continua distante das demais em tempo total.

(ii) **Balanceamento** (Figura 94b): **DLS** mantém $\bar{U} \approx 1$ em todo o intervalo, enquanto as demais aumentam gradualmente a razão de balanceamento com P (linhas quase

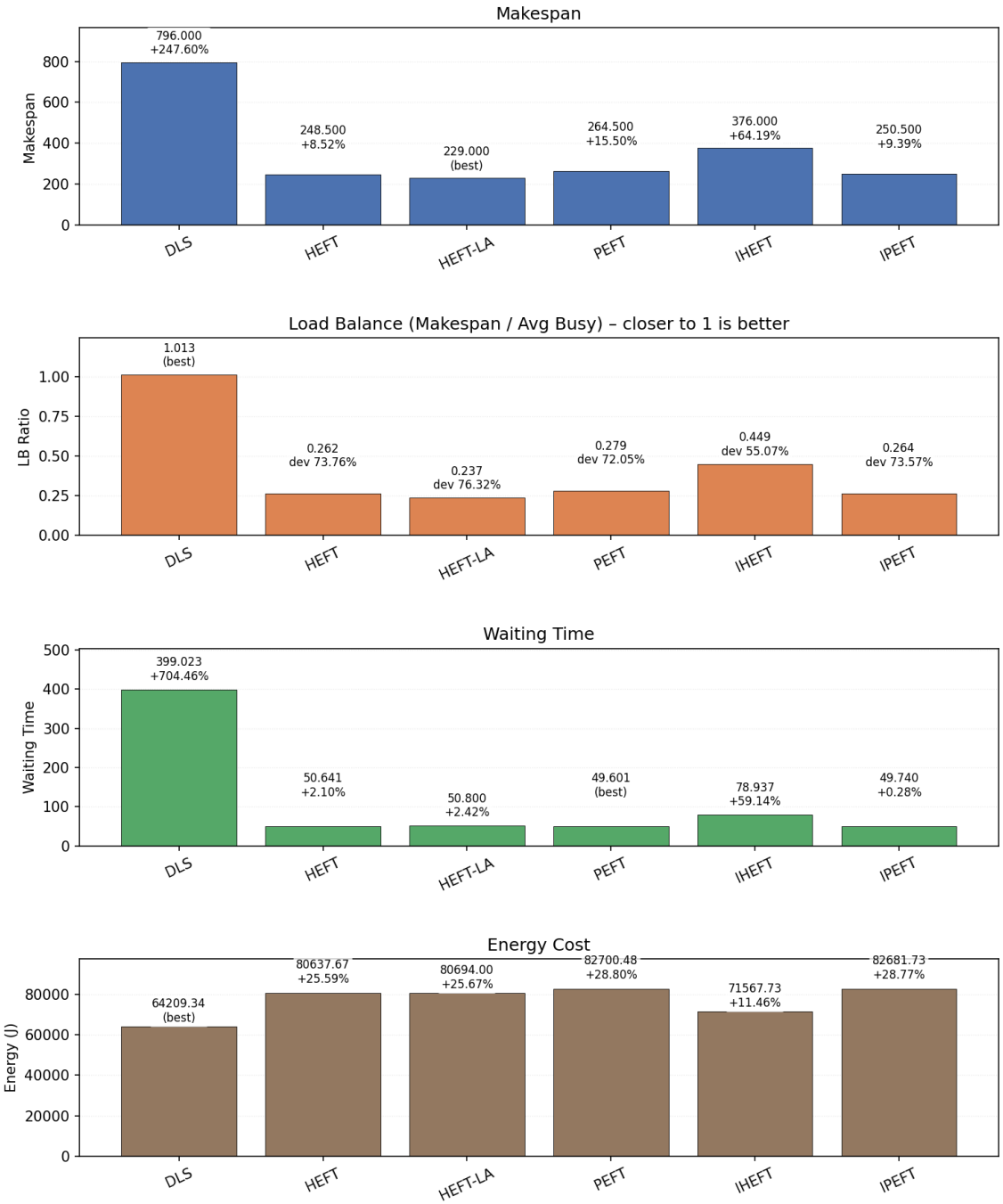


Figura 87 – Métricas (MS, $\bar{U}$, $\overline{\mathrm{WT}}$, E_{total}) – cenário 4, P=32.

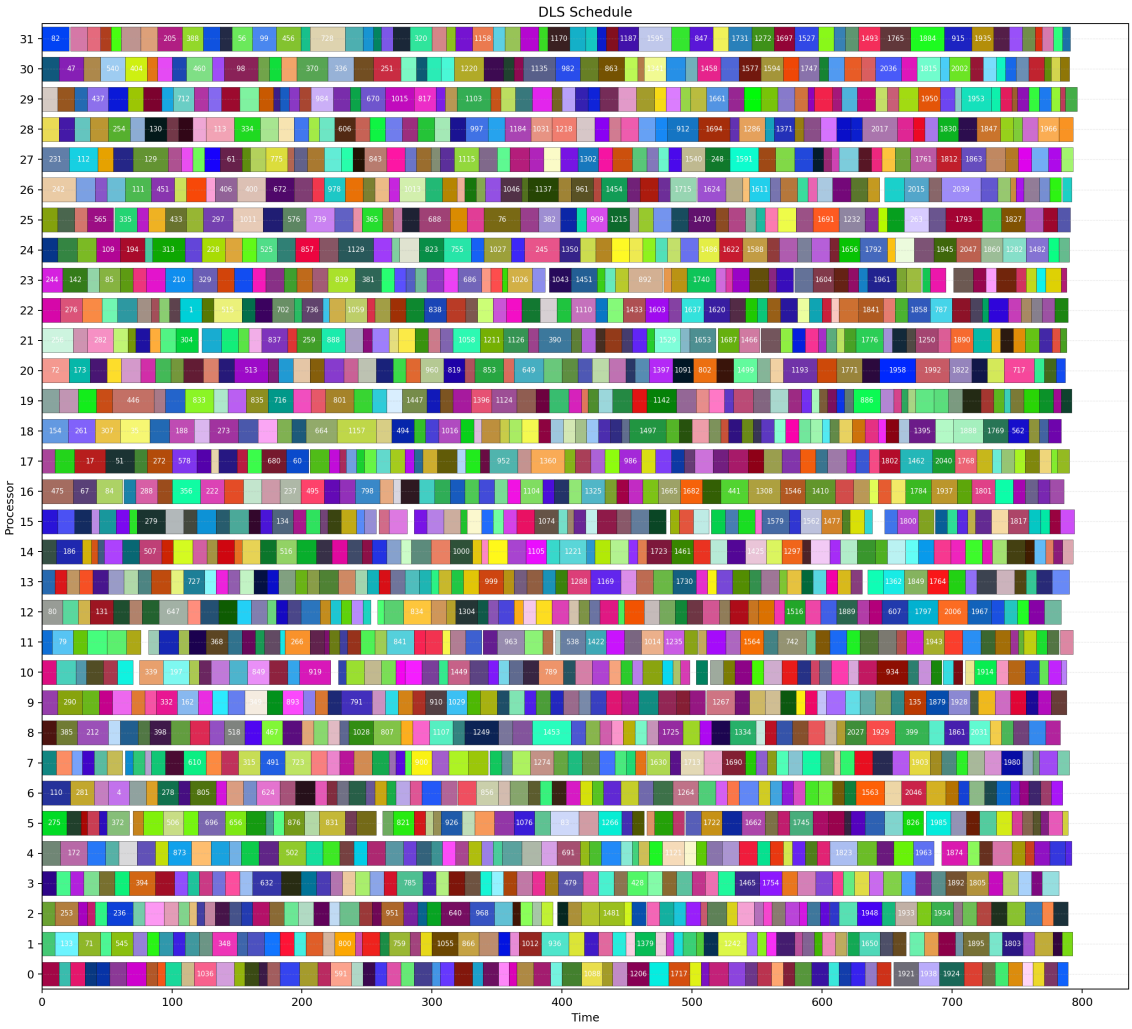


Figura 88 – Gantt – DLS, cenário 4, P=32.

lineares), porém ainda $\bar{U} \ll 1$–um indicativo de gargalos de precedência/comunicação que limitam o paralelismo efetivo quando N é grande.

(iii) **Espera** (Figura 94c): **PEFT** e **IPEFT** e **HEFT** e **HEFT-LA** mantêm $\overline{\mathrm{WT}}$ baixa e estável ao longo de P. **IHEFT** tem filas maiores, e **DLS** apresenta a maior espera (ainda que decrescente com P), coerente com seu escalonamento mais "equilibrado", porém mais longo.

(iv) **Energia** (Figura 94d): todas as políticas reduzem E_{total} ao aumentar P, mas aqui o **DLS** sobressai como o mais eficiente energeticamente em toda a faixa. Entre as heurísticas com melhor MS (PEFT/IPEFT/HEFT/HEFT-LA), **IHEFT** costuma ser o mais econômico, sugerindo um *trade-off* tempo–energia mais favorável que o das demais inserções.

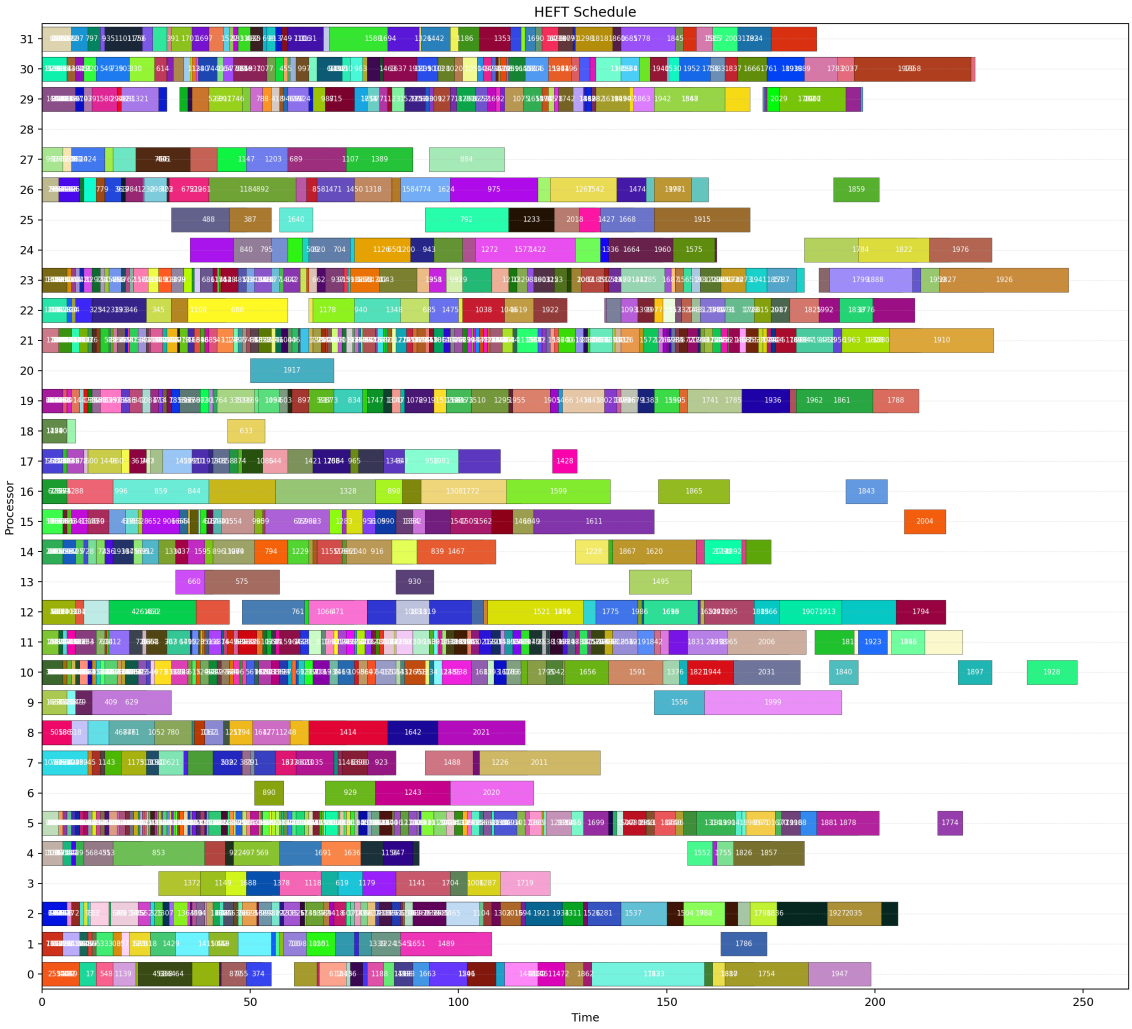


Figura 89 – Gantt – HEFT, cenário 4, P=32.

4.5.1.0.2 Leituras cruzadas (sem comparar valores absolutos).

Em linha com os cenários anteriores, o **padrão se repete**: heurísticas com inserção priorizam caminho crítico e minimizam MS, aceitando menor $\bar{U}$; **DLS** maximiza $\bar{U}$ mas paga em tempo; **IHEFT** é o meio-termo mais *energético*. No cenário 4 (muito mais tarefas), esses efeitos ficam *mais visíveis*: ganhos de MS ao ir de $P = 16$ para $P = 32$ são modestos para as melhores políticas (limites de paralelismo do DAG), enquanto o balanceamento cresce quase linearmente–útil para utilização, mas com retorno decrescente em tempo. Em sistemas onde a prioridade é **prazo**, PEFT/IPEFT/HEFT-LA seguem recomendáveis; em metas de **eficiência energética**, **IHEFT** e, em última instância, **DLS** podem ser considerados, desde que haja folga temporal.

4.6 Cenário 5: $N = 4096$ tarefas

O DAG ilustrativo do cenário 5 está mostrado na Figura 95.

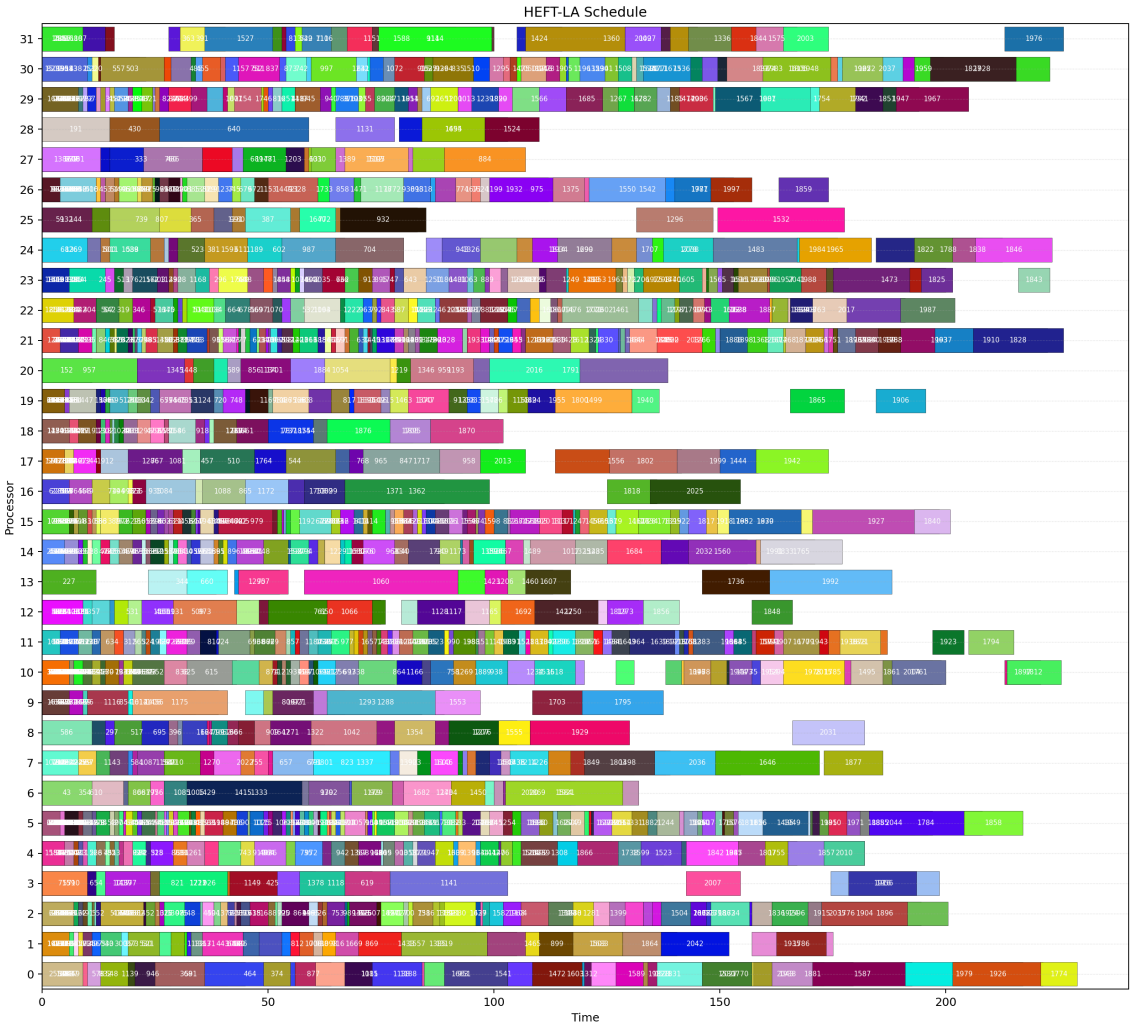


Figura 90 – Gantt – HEFT-LA, cenário 4, P=32.

4.6.1 Configuração com $P = 8$

4.6.1.1 Análise P=8

Em Figura 96, o padrão geral dos cenários anteriores se preserva: **PEFT** e **IPEFT** entregam os menores MS e $\overline{\text{WT}}$, com **HEFT** e **HEFT-LA** logo atrás; **DLS** mantém $\bar{U} \approx 1$ porém com MS muito alto. As diferenças específicas deste cenário são: (i) o **IHEFT** ganha ainda menos ao tentar inserções com P=8; os Gantts (Figura 101) exibem caudas tardias em PEs menos velozes; (ii) **PEFT** e **IPEFT** e **HEFT** e **HEFT-LA** ficam muito próximos em E_{total}, enquanto o **DLS** tende a consumo levemente menor por menor paralelismo efetivo, à custa de tempo bem pior. Nos Gantts, vê-se a *concentração* do caminho crítico em **PEFT** e **IPEFT** (Figura 100, Figura 102), uma distribuição mais espalhada em **HEFT** e **HEFT-LA** (Figura 98, Figura 99) e forte fragmentação em **DLS** (Figura 97).



Figura 91 – Gantt – PEFT, cenário 4, P=32.

4.6.2 Configuração com $P = 16$

4.6.2.1 Análise P=16

Makespan. O **PEFT** volta a ficar ligeiramente à frente (MS≈265,5) com **IPEFT** muito próximo (≈ 269,5); **HEFT** segue logo atrás (≈ 278,5) enquanto **HEFT-LA** não traz ganho aqui (≈ 313), ficando pior que o HEFT "puro". **IHEFT** é bem mais lento (≈ 440) e **DLS** permanece fora de escala (≈ 3758); cf. Figura 103.

Balanceamento. Como nos demais cenários, **DLS** é quase perfeito ($\bar{U} \approx 1{,}00$), e as demais heurísticas ficam com $\bar{U} \ll 1$. Entre as não-DLS, o **IHEFT** exibe o maior $\bar{U}$ (≈ 0,107), mas mantém MS alto. Os Gantts confirmam: **PEFT** e **IPEFT** concentram o caminho crítico (Figura 107, Figura 109); **HEFT-LA** fragmenta mais e cria cauda (Figura 106); **DLS** distribui amplamente e alonga a linha do tempo (Figura 104).

Espera e energia. Em $\overline{\mathrm{WT}}$, **IPEFT** é o melhor (≈ 51,4) com **PEFT** praticamente empatado; **HEFT** e **HEFT-LA** mantêm filas moderadas, **IHEFT** aumenta a espera e **DLS** segue muito alto. Em E_{total}, **DLS** minimiza o consumo (troca tempo por



Figura 92 – Gantt – IHEFT, cenário 4, P=32.

energia). Entre os métodos com bom MS, **IHEFT** consome menos que **PEFT** e **IPEFT**, evidenciando novamente o *trade-off* tempo–energia.

4.6.3 Configuração com $P = 32$

4.6.3.1 Análise P=32

Em Figura 110, **DLS** apresenta $\bar{U} \approx 1$ porém com MS e $\overline{\mathrm{WT}}$ bem maiores que os demais; {**PEFT**, **IPEFT**, **HEFT**, **HEFT-LA**} permanecem agrupados com MS baixo e filas curtas. A nuance é que, com 32 PEs, as diferenças entre **HEFT-LA**, **HEFT** e **PEFT** e **IPEFT** ficam ainda menores (trocas de liderança por margens pequenas), enquanto o **IHEFT** degrada relativamente em tempo e espera, mantendo leve vantagem energética.

Os Gantts de **PEFT** e **IPEFT** (Figura 114, Figura 116) mostram maior ociosidade em PEs periféricos e *faixas longas* nos PEs críticos; **HEFT** e **HEFT-LA** (Figura 112, Figura 113) preenchem lacunas de modo competitivo; **DLS** (Figura 111) distribui intensamente e alonga a linha do tempo; **IHEFT** (Figura 115) evidencia caudas por mapeamentos



Figura 93 – Gantt – IPEFT, cenário 4, P=32.

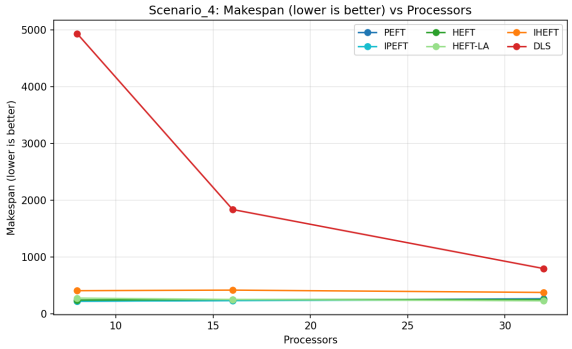
energeticamente favoráveis.

4.6.3.2 Síntese P=32

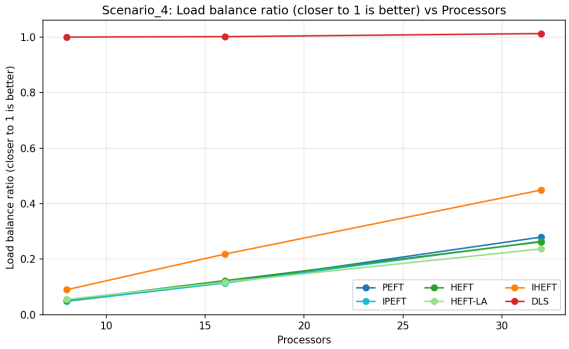
(i) As heurísticas "rápidas" convergem — **HEFT-LA**, **HEFT**, **PEFT** e **IPEFT** ficam praticamente empatadas em MS e $\overline{\text{WT}}$; (ii) **IHEFT** perde terreno em tempo, mantendo leve economia de energia; (iii) Com N elevado, essas heurísticas exibem $\bar{U}$ maior que nos cenários menores (melhor preenchimento), mas ainda abaixo de 1; quem se aproxima de 1 é o **DLS**, que paga em MS (iv) **DLS** segue como *outlier*: ótimo $\bar{U}$, pior tempo.

4.6.4 Síntese do cenário 5 (N=4096)

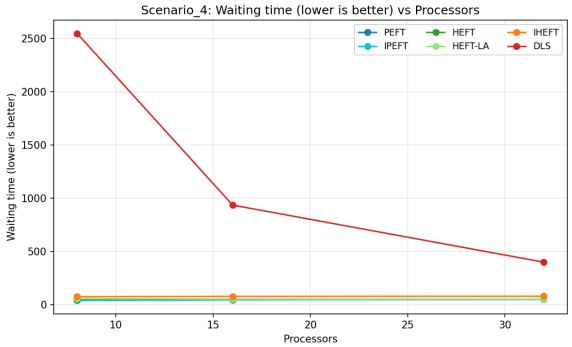
Principais achados. (Figura 117a) **PEFT** e **IPEFT** sustentam o menor MS com P=8, 16; com P=32, **HEFT** e **HEFT-LA** tornam-se virtualmente indistinguíveis das variantes com inserção, sinalizando *saturação por precedência/comunicação*. (Figura 117b) **DLS** mantém $\bar{U} \approx 1$ em toda a faixa, enquanto as heurísticas rápidas aumentam gra-



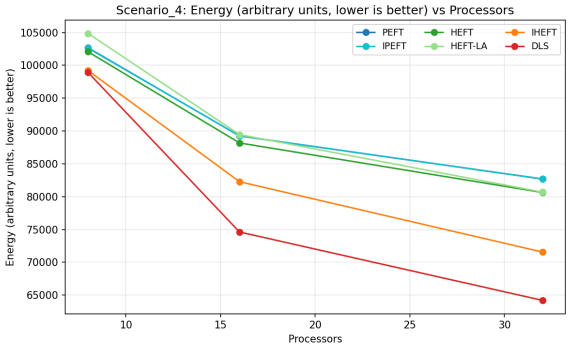
(a) Makespan (↓ melhor).



(b) Razão de balanceamento de carga (≈ 1 é melhor).



(c) Tempo médio de espera $\overline{\mathrm{WT}}$ (↓ melhor).



(d) Energia total E_{total} (↓ melhor).

Figura 94 – Síntese visual do cenário 4: evolução das métricas com $P \in \{8, 16, 32\}$.

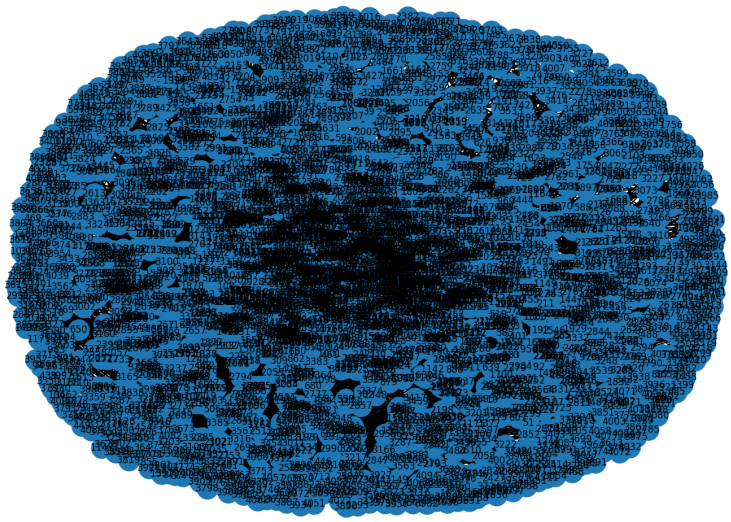


Figura 95 – DAG ilustrativo do cenário 5 (N=4096).

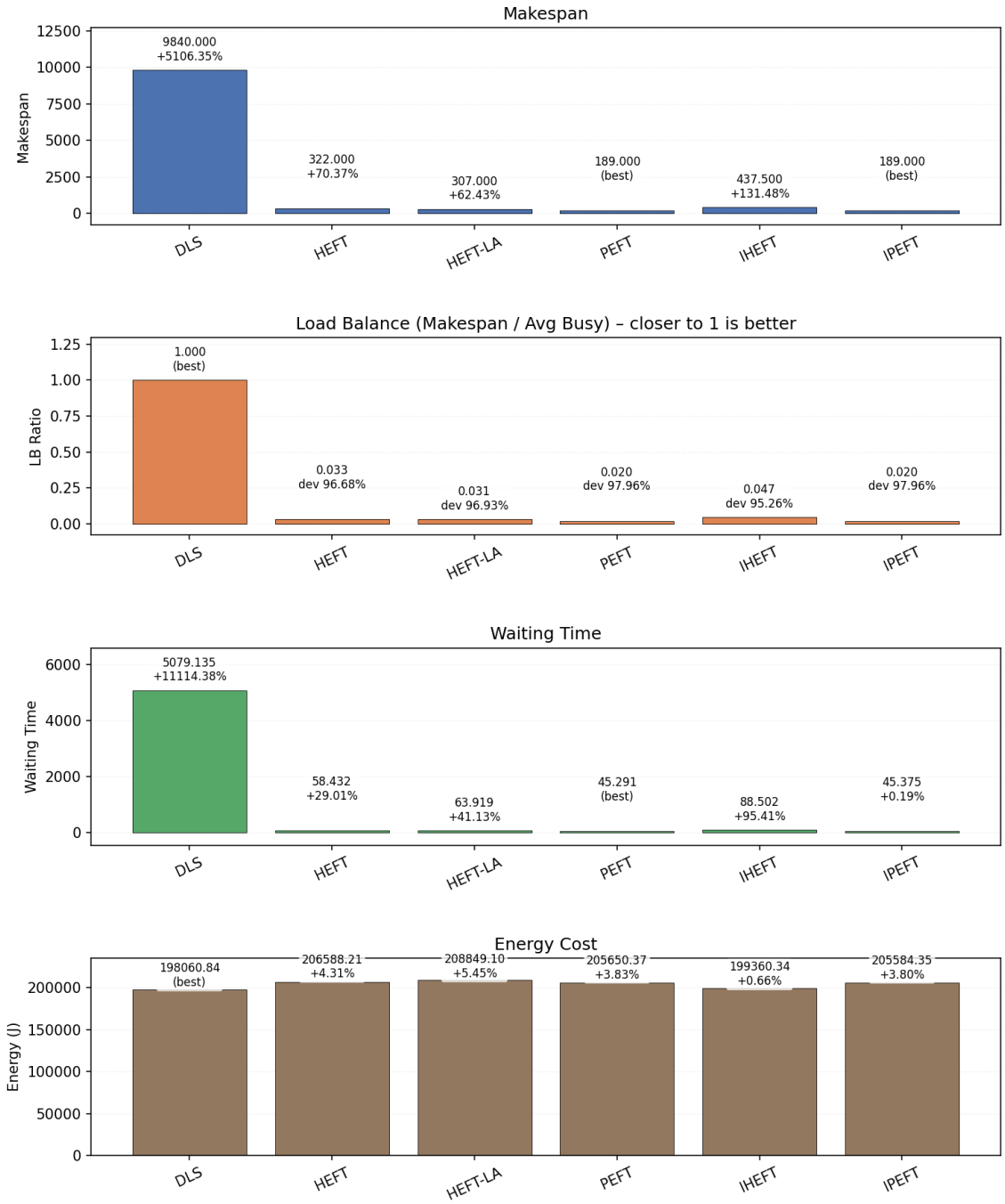


Figura 96 – Métricas (MS, $\bar{U}$, $\overline{\mathrm{WT}}$, E_{total}) – cenário 5, P=8.

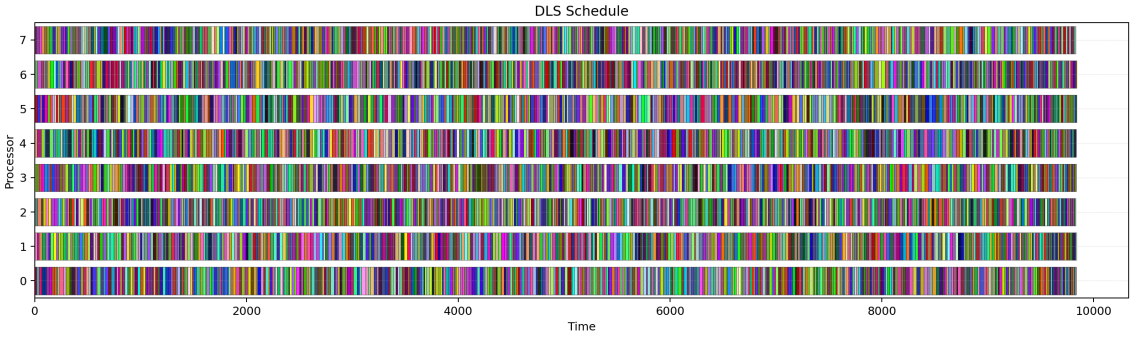


Figura 97 – Gantt – DLS, cenário 5, P=8.

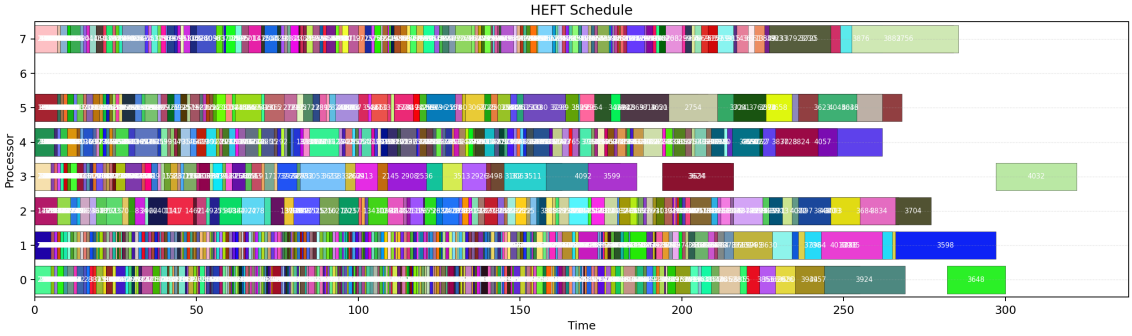


Figura 98 – Gantt – HEFT, cenário 5, P=8.

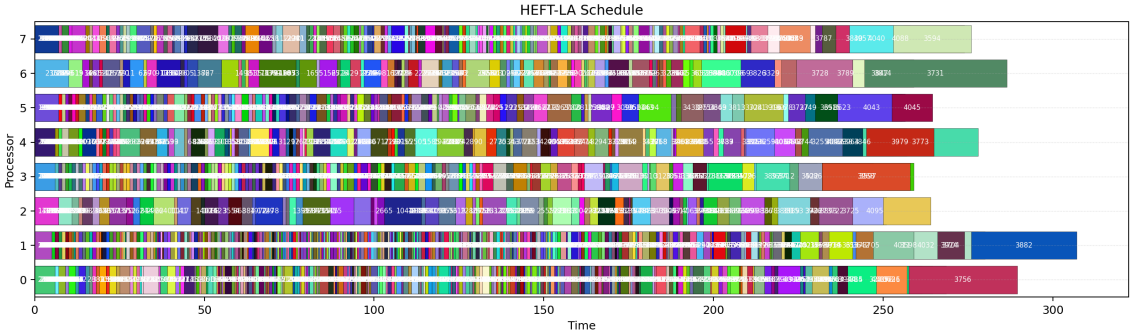


Figura 99 – Gantt – HEFT-LA, cenário 5, P=8.

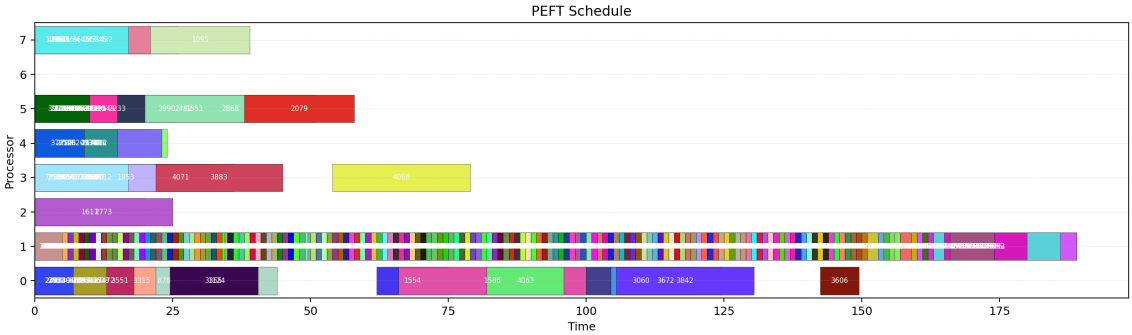


Figura 100 – Gantt – PEFT, cenário 5, P=8.

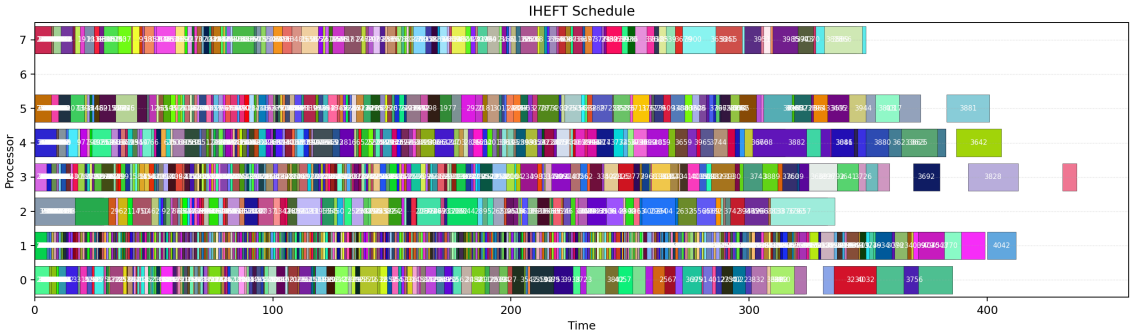


Figura 101 – Gantt – IHEFT, cenário 5, P=8.

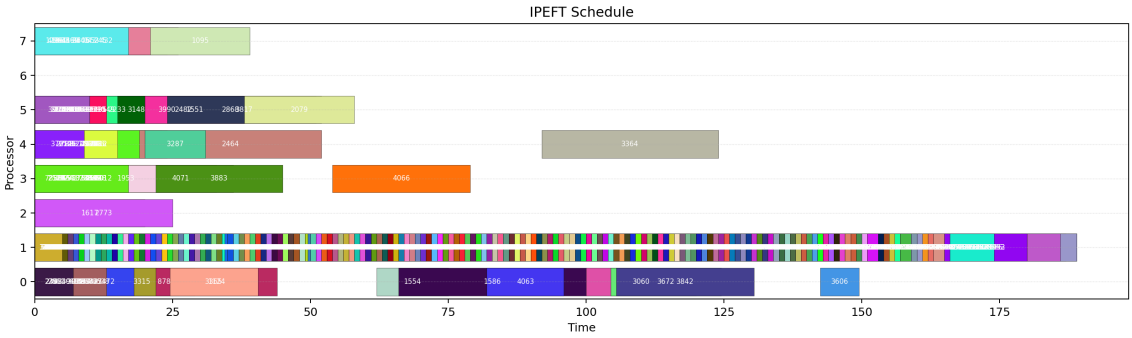


Figura 102 – Gantt – IPEFT, cenário 5, P=8.

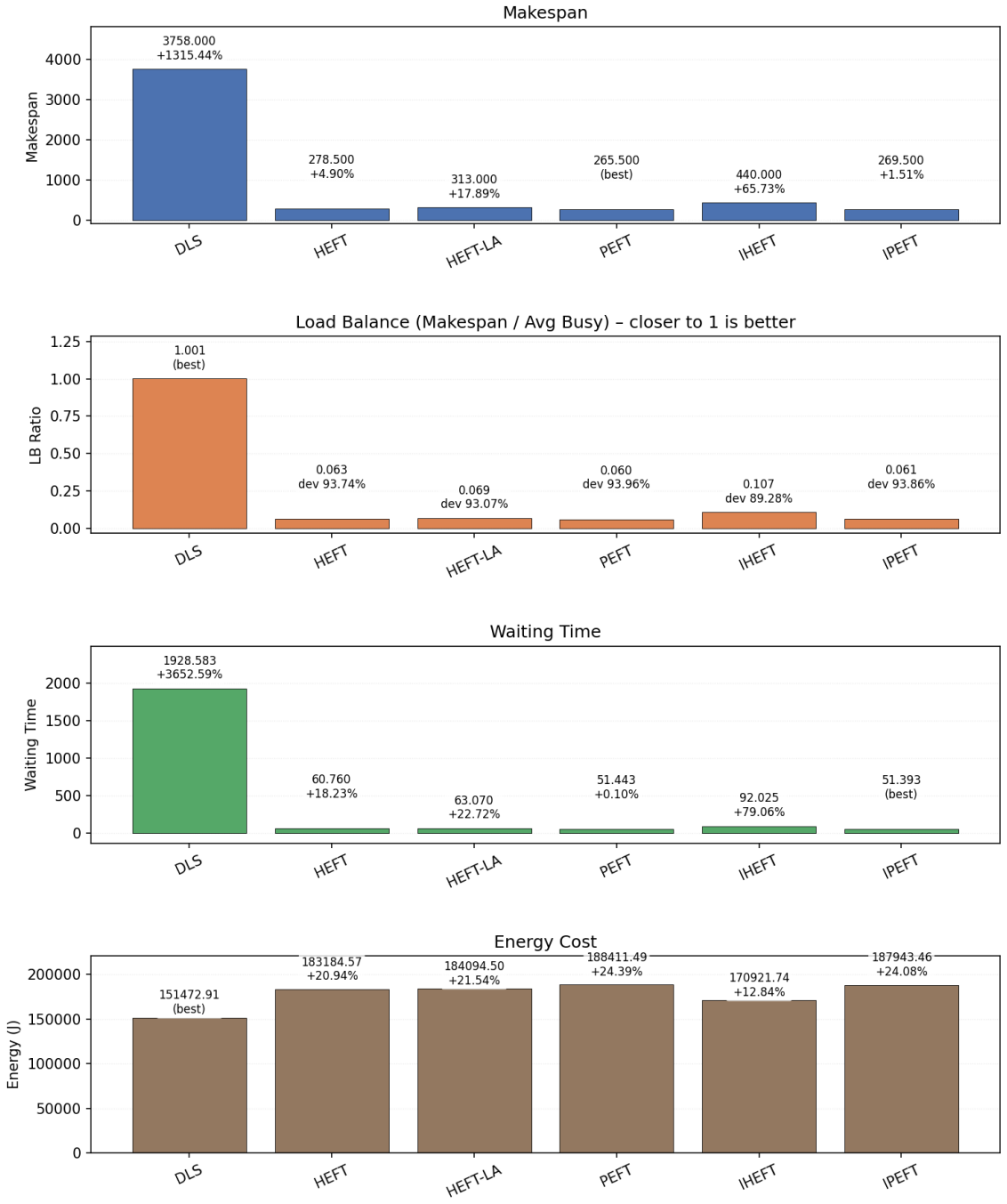


Figura 103 – Métricas (MS, $\bar{U}$, $\overline{\mathrm{WT}}$, E_{total}) – cenário 5, P=16.



Figura 104 – Gantt – DLS, cenário 5, P=16.

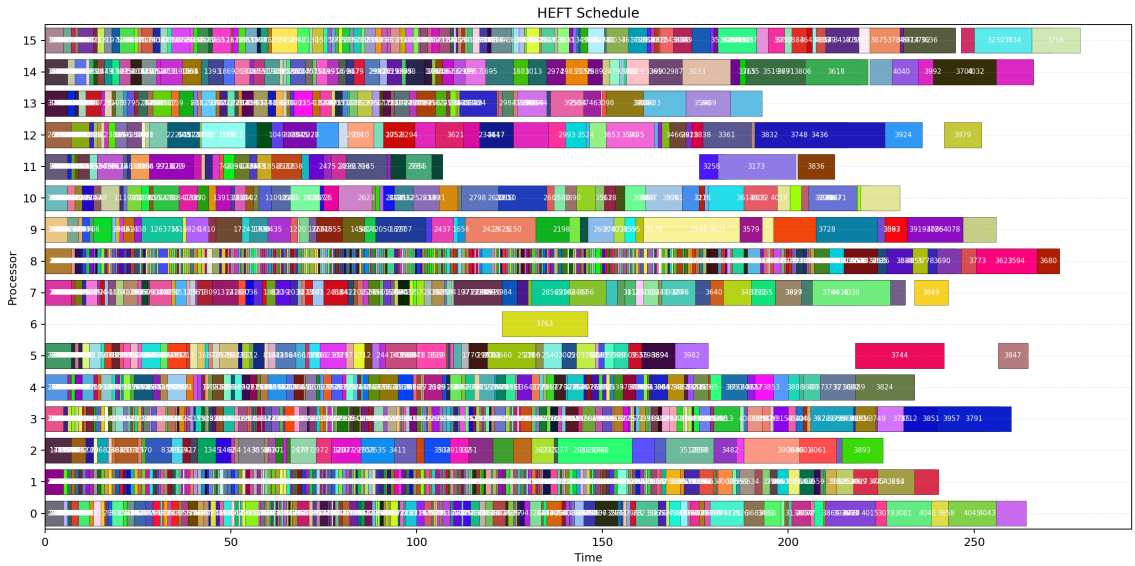


Figura 105 – Gantt – HEFT, cenário 5, P=16.

dualmente o $\bar{U}$ com P sem atingir 1 — reflexo de concentração do caminho crítico. (Figura 117c) **IPEFT** e **PEFT** exibem as menores filas, seguidos de **HEFT** e **HEFT-LA**; **IHEFT** e **DLS** mantêm esperas maiores. (Figura 117d) Em energia, **DLS** costuma ser o mais econômico, com **IHEFT** frequentemente abaixo das heurísticas com melhor MS — reforçando o *trade-off* tempo–energia.

Leituras cruzadas (sem comparar valores absolutos). O padrão estratégico observado nos demais cenários se reforça no limite de N grande: (1) Para *time-to-train*, priorizar **PEFT** e **IPEFT** (e **HEFT** e **HEFT-LA** com muitos PEs) é superior a buscar balanceamento global; (2) Para *eficiência energética*, **DLS** e **IHEFT** são alternativas com penalidade de tempo; (3) O ganho de P=16 → 32 satura: o gargalo dominante é o

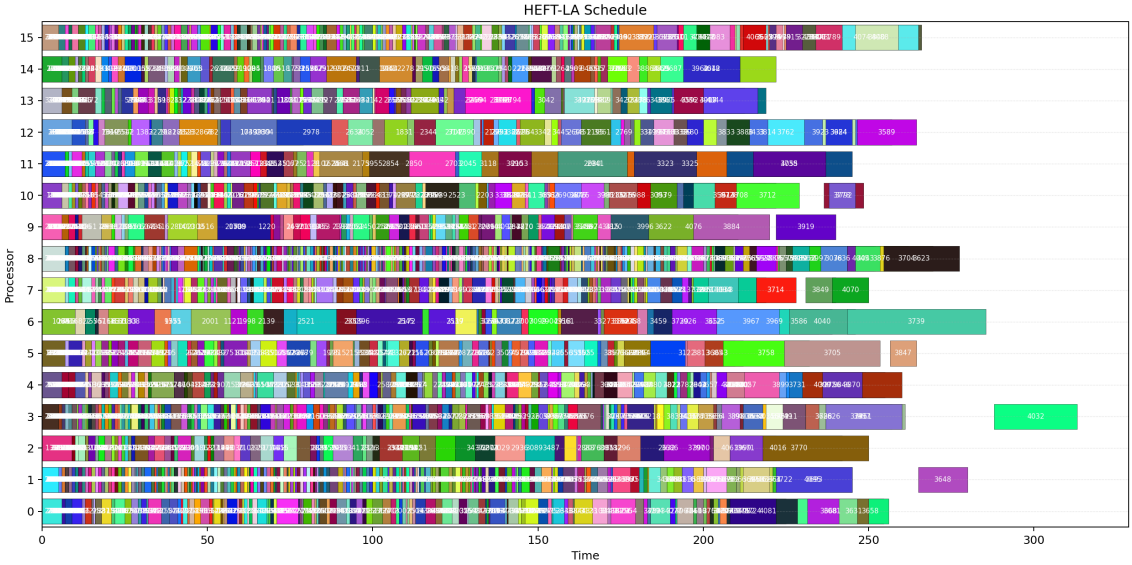


Figura 106 – Gantt – HEFT-LA, cenário 5, P=16.

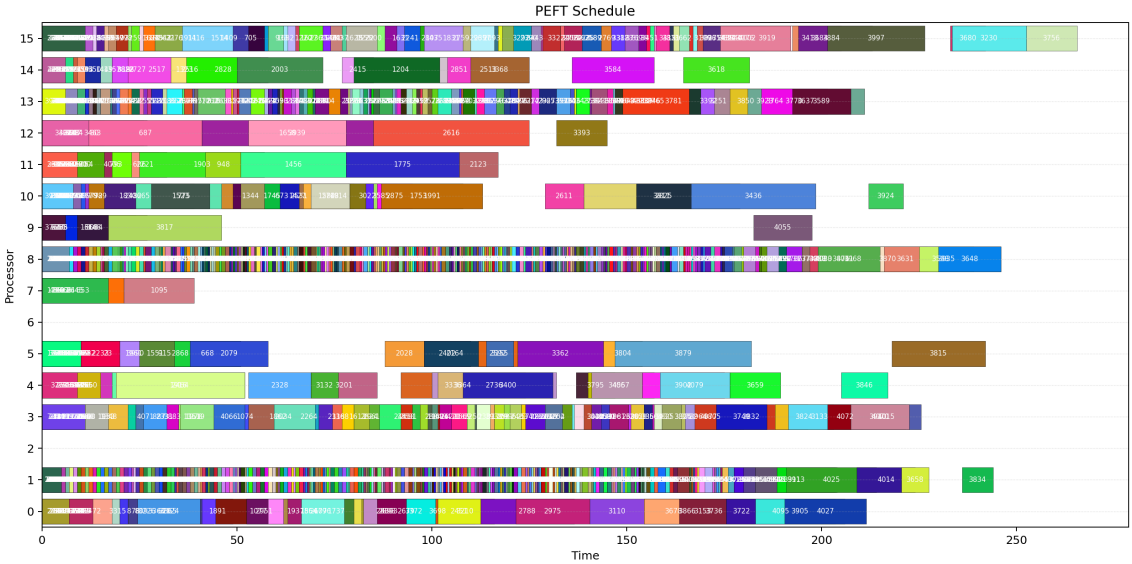


Figura 107 – Gantt – PEFT, cenário 5, P=16.

caminho crítico e os **custos de comunicação**, não a falta de PEs.

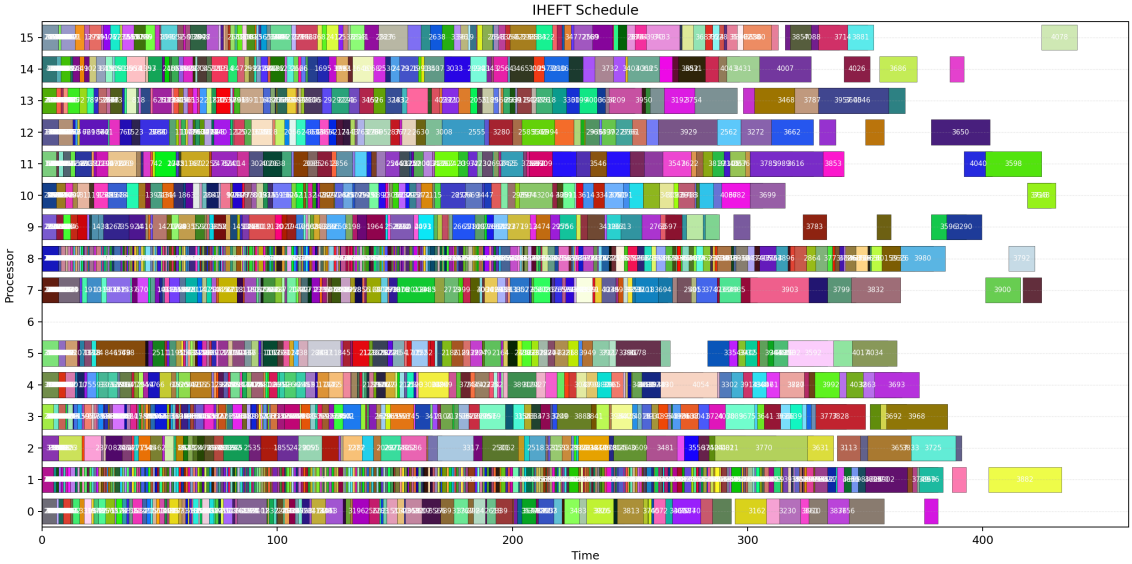


Figura 108 – Gantt – IHEFT, cenário 5, P=16.

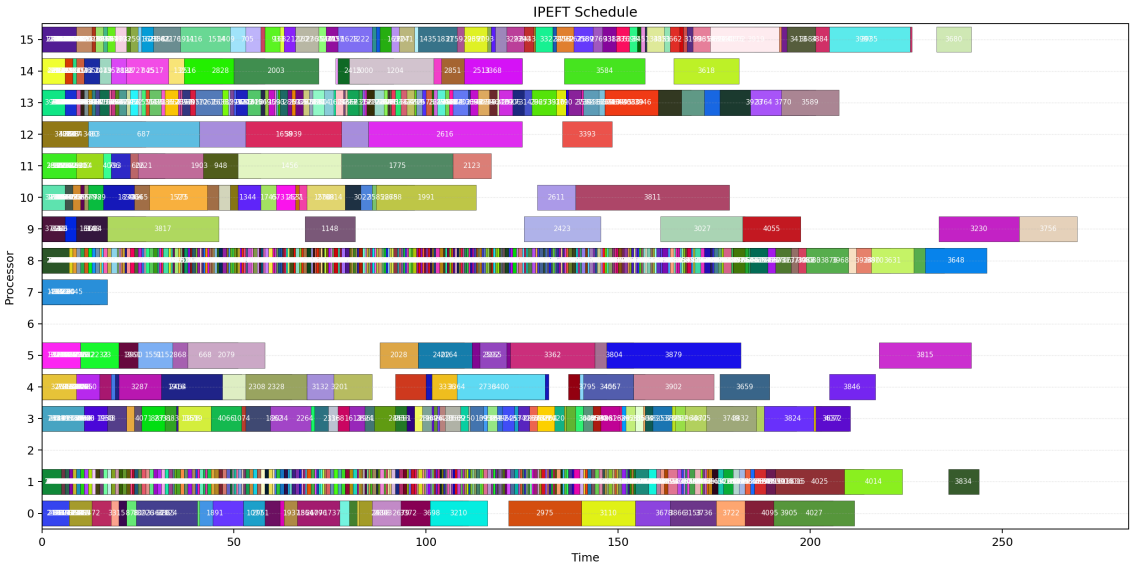


Figura 109 – Gantt – IPEFT, cenário 5, P=16.

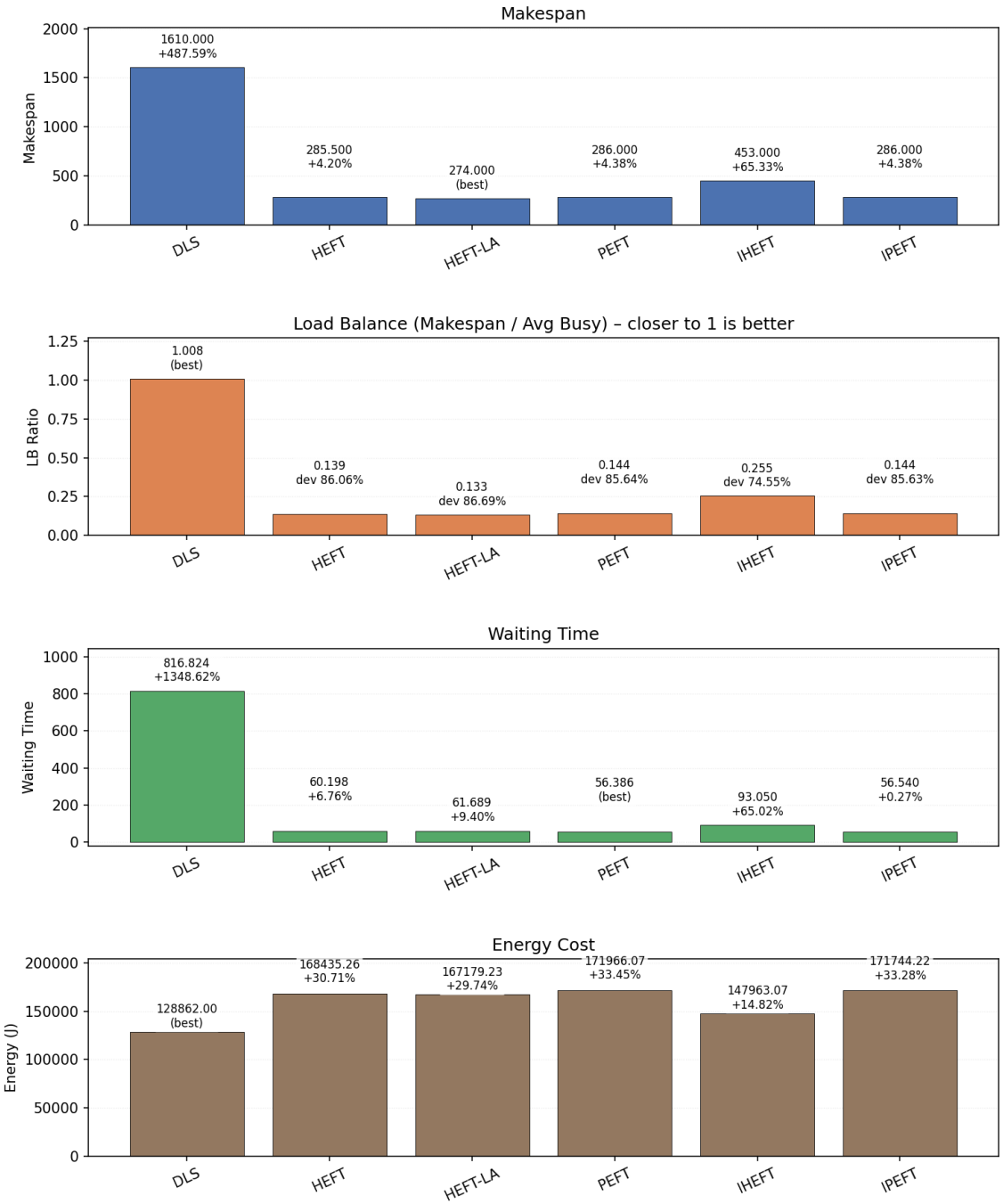


Figura 110 – Métricas (MS, $\bar{U}$, $\overline{\mathrm{WT}}$, E_{total}) – cenário 5, P=32.

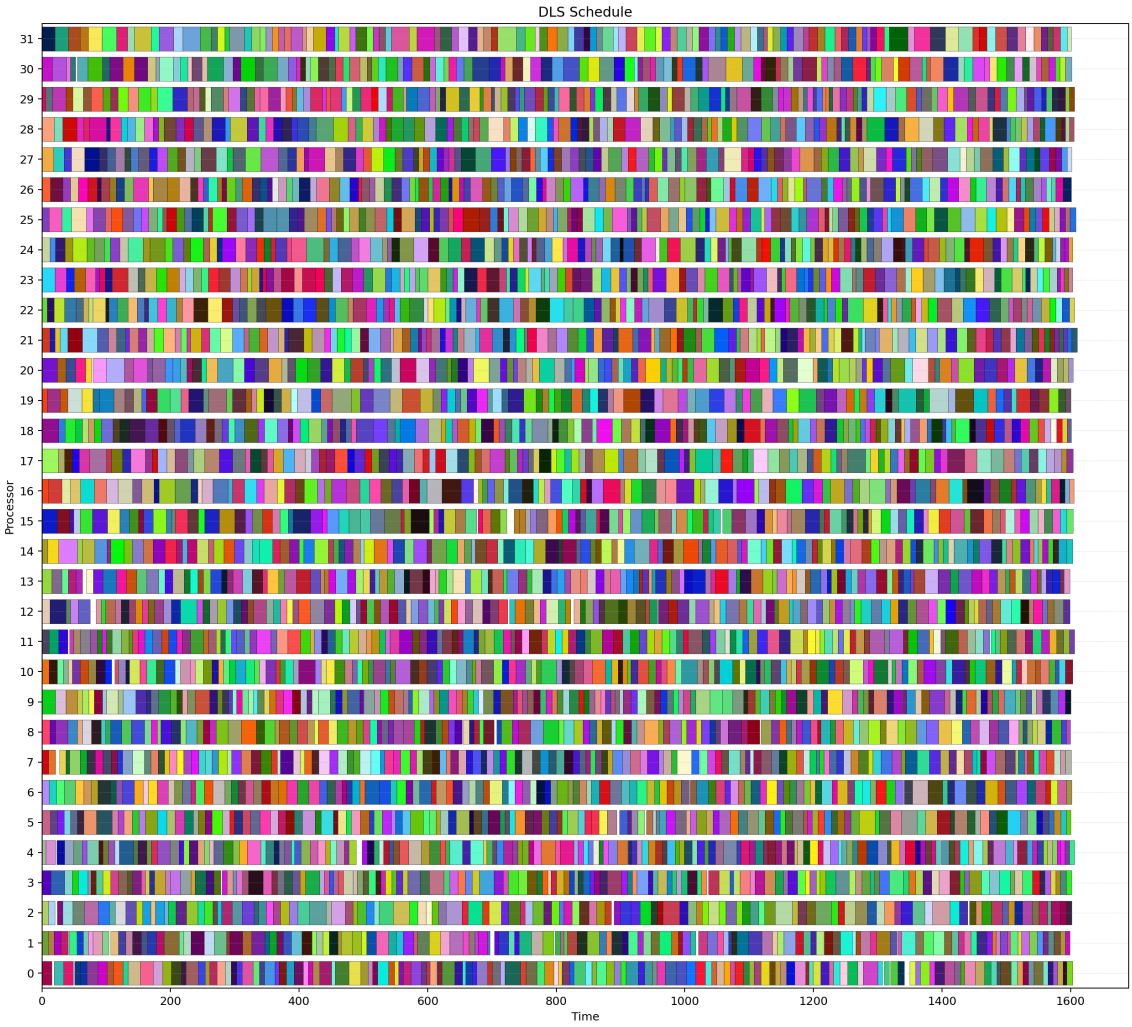


Figura 111 – Gantt – DLS, cenário 5, P=32.

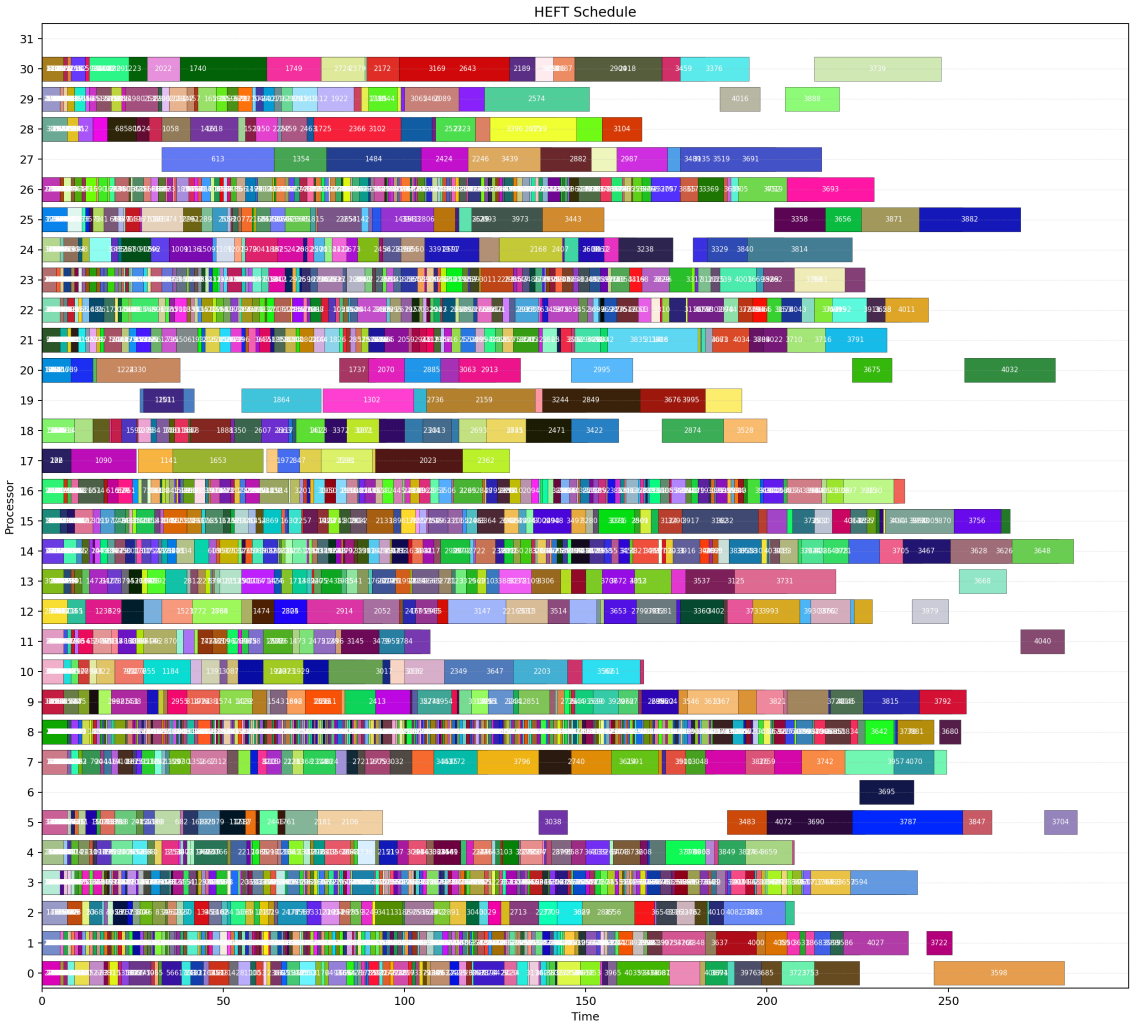


Figura 112 – Gantt – HEFT, cenário 5, P=32.

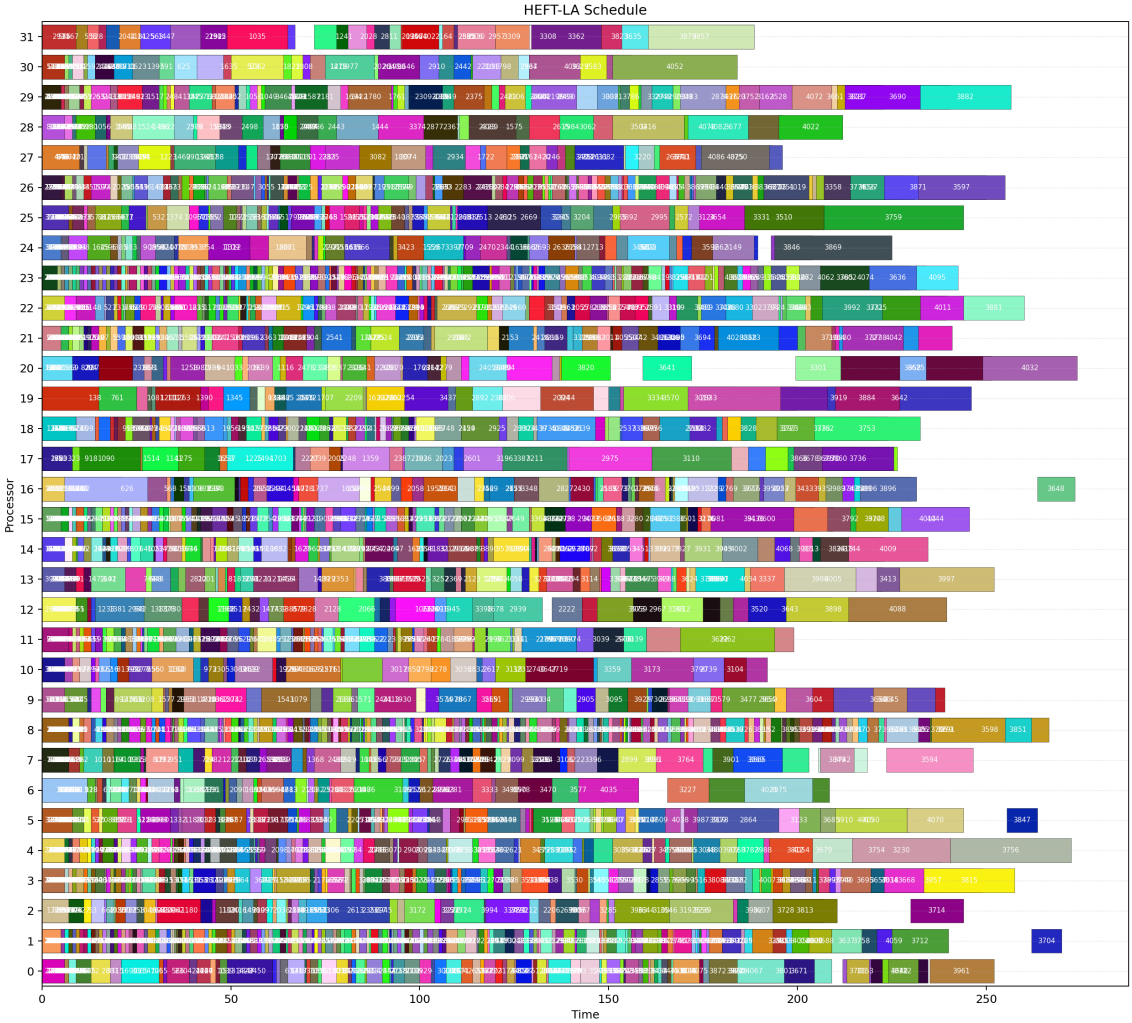


Figura 113 – Gantt – HEFT-LA, cenário 5, P=32.



Figura 114 – Gantt – PEFT, cenário 5, P=32.

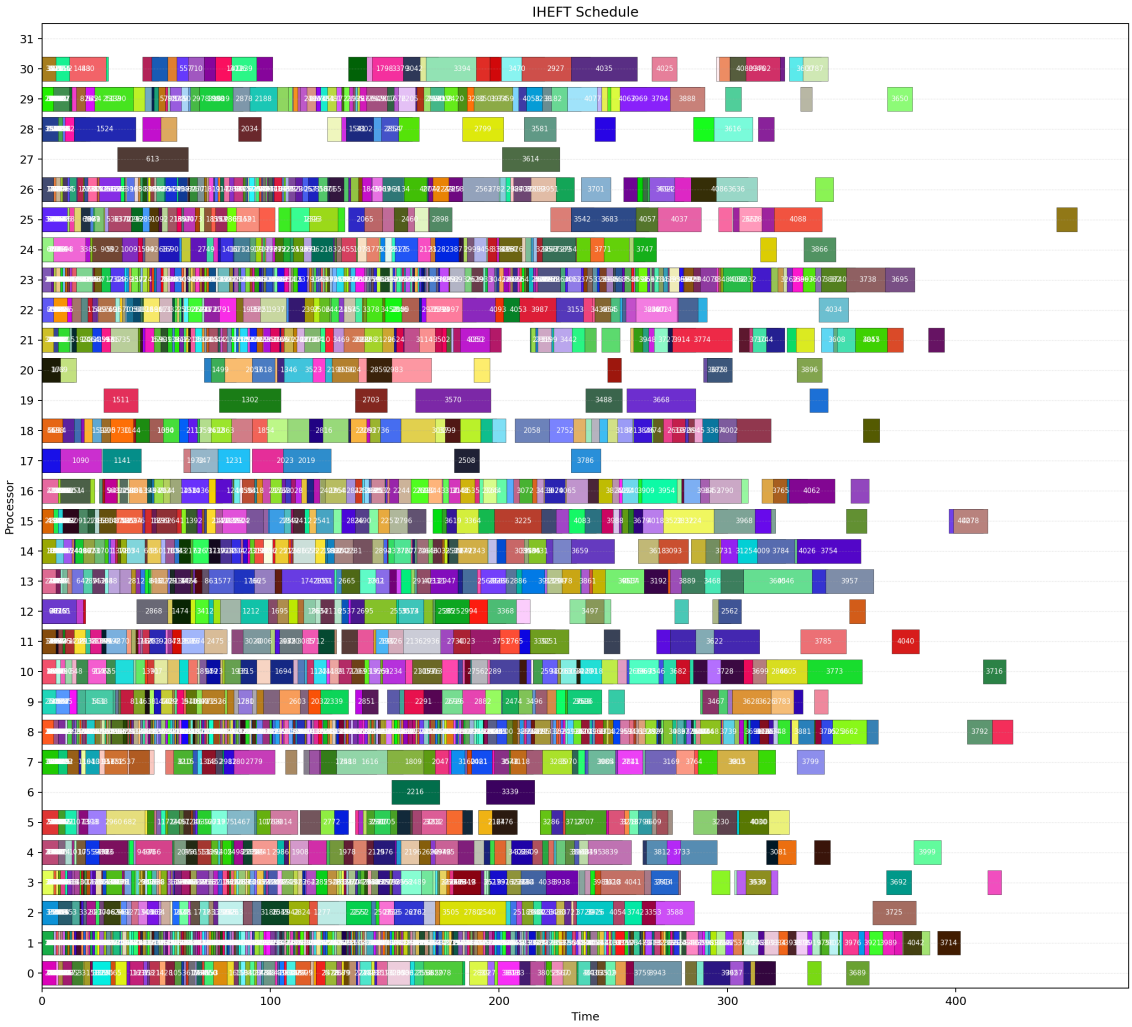
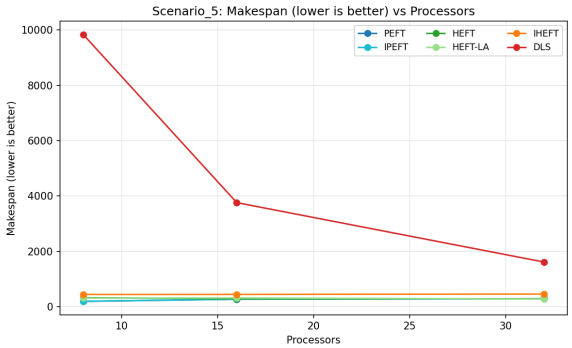


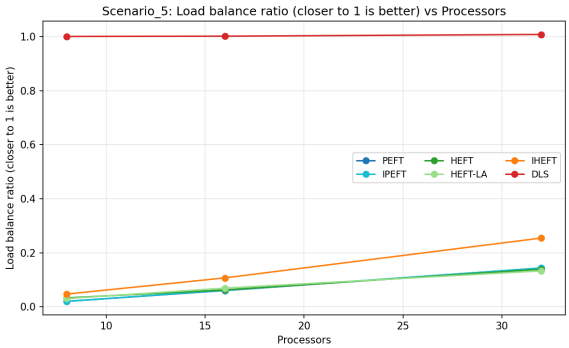
Figura 115 – Gantt – IHEFT, cenário 5, P=32.



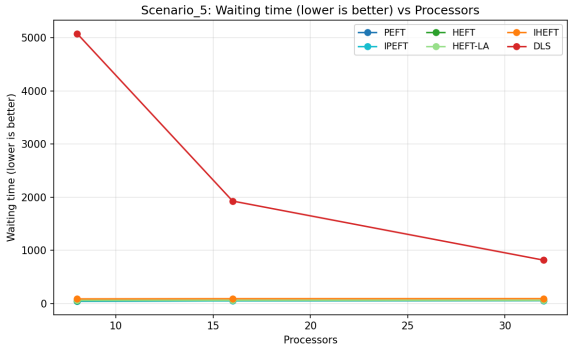
Figura 116 – Gantt – IPEFT, cenário 5, P=32.



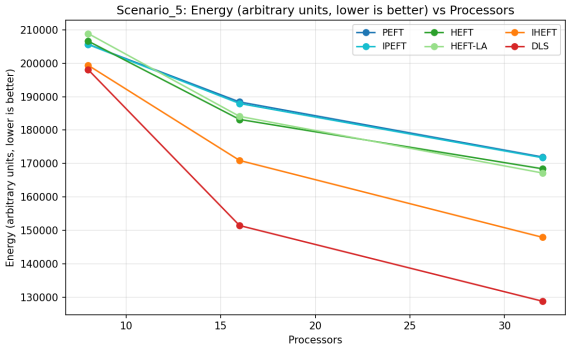
(a) Makespan (↓ melhor).



(b) Razão de balanceamento de carga (≈ 1 é melhor).



(c) Tempo médio de espera $\overline{\mathrm{WT}}$ (↓ melhor).



(d) Energia total E_{total} (↓ melhor).

Figura 117 – Síntese visual do cenário 5: evolução das métricas com $P \in \{8, 16, 32\}$.

5 Conclusão

Este trabalho comparou, de forma sistemática e reprodutível, seis heurísticas clássicas e modernas de escalonamento de DAGs em ambientes heterogêneos (DLS, HEFT, HEFT-LA, PEFT, IHEFT e IPEFT), em cenários que espelham cargas contemporâneas de *machine learning*/LLMs distribuídos e sua pressão por desempenho e eficiência energética (YANG et al., 2024; YAN et al., 2024; YAN et al., 2025; STRUBELL; GANESH; MCCALLUM, 2019; SCHWARTZ et al., 2020; PATTERSON et al., 2021; HENDERSON et al., 2020; LACOSTE et al., 2019). As evidências empíricas coletadas ao longo dos cenários permitem afirmar, com base nas formulações originais, por que cada heurística se comportou como observado.

(1) Predominância das heurísticas do "núcleo HEFT" em tempo. O *HEFT* executa priorização por *upward rank* e seleciona o processador que minimiza o *Earliest Finish Time* (EFT) com política de inserção, o que, em ambientes heterogêneos, produz cronogramas curtos com custo computacional baixo (TOPCUOGLU; HARIRI; WU, 2002). O *PEFT* preserva essa simplicidade e acrescenta uma *Optimistic Cost Table* (OCT) que antecipa custos futuros, refinando a ordem e a escolha do processador (ARABNEJAD; BARBOSA, 2013). O *HEFT-LA* aplica *look-ahead* explícito ao avaliar o impacto da alocação atual sobre os descendentes, evitando decisões localmente gananciosas (BITTENCOURT; SAKELLARIOU; MADEIRA, 2010). Por fim, o *IPEFT* combina EFT com a noção de nós/critérios críticos (PCT/CNCT), escolhendo o par tarefa–processador que minimiza o efeito sobre os caminhos descendentes mais sensíveis (ZHOU et al., 2016). Em conjunto, "lista + inserção + antevisão" explica por que HEFT, HEFT-LA, PEFT e IPEFT formam, de modo geral, o pelotão de frente em *makespan* e tempo de espera nos cenários testados.

(2) Papel do IHEFT. A família *IHEFT* ajusta o ranqueamento/seleção para dar mais peso a caminhos críticos e lacunas no cronograma, mantendo o espírito "lista + inserção" do HEFT (ALEBRAHIM; AHMAD, 2016). Nos experimentos, essa variação preservou boa eficiência energética (menor ociosidade em PEs custosos) ao custo de, por vezes, ficar atrás das variantes com *look-ahead* explícito em *makespan* quando a comunicação se torna dominante.

(3) Por que o DLS é o *outlier* – e por que escalou quase linearmente com mais processadores. O *DLS* seleciona iterativamente o par tarefa–processador de maior *Dynamic Level* (DL), calculado a partir do nível estático (*Static Level*, que privilegia proximidade do caminho crítico) menos o *Earliest Execution Start Time* (*EEST*) corrente (SIH; LEE, 1993). Essa regra, ao depender do *EEST* de *todos* os PEs, "preenche

o primeiro buraco" disponível, garantindo *load balance* muito alto mesmo sem a política de inserção/antevisão do HEFT. Em DAGs grandes com alto paralelismo, o aumento de PEs reduz simultaneamente vários *EEST*s, permitindo ao DLS escalar o paralelismo de modo quase linear – fenômeno que observamos experimentalmente e que decorre diretamente de sua formulação (SIH; LEE, 1993). O preço dessa estratégia é que, na presença de dependências fortes/CCR maiores, o DLS pode alongar o caminho crítico em relação às variantes com *look-ahead* (HAGRAS; JANEČEK, 2003).

(4) Convergência com muitos PEs e muitas tarefas. Quando o paralelismo intrínseco é elevado (cenários "grandes") e os recursos são abundantes, as heurísticas rápidas convergem: HEFT-LA, HEFT, PEFT e IPEFT ficam praticamente empatadas em *makespan* e tempo de espera, e o *load balance* dessas heurísticas aproxima-se de valores muito bons. Isso é coerente com seus desenhos, pois todas conseguem manter o caminho crítico continuamente "alimentado" graças à inserção e à antevisão (TOPCUOGLU; HARIRI; WU, 2002; ARABNEJAD; BARBOSA, 2013; BITTENCOURT; SAKELLARIOU; MADEIRA, 2010; ZHOU et al., 2016).

(5) Energia e tempo de espera. O consumo energético agregado acompanhou majoritariamente o *makespan*, com exceções quando a política evitou concentrar tarefas em PEs de alto custo. Esse resultado está alinhado com estudos que relacionam tempo de execução e custo energético em cargas de ML, reforçando a necessidade de reportar e otimizar ambas as dimensões (STRUBELL; GANESH; MCCALLUM, 2019; SCHWARTZ et al., 2020; PATTERSON et al., 2021; HENDERSON et al., 2020; LACOSTE et al., 2019).

Contribuições e impacto

- **Base comparativa unificada.** Implementamos fielmente as formulações originais e avaliamos seis heurísticas sob métricas relevantes (makespan, *load balance*, tempo de espera e energia), oferecendo uma linha de base clara para escalonamento de DAGs em hardware heterogêneo (TOPCUOGLU; HARIRI; WU, 2002; ARABNEJAD; BARBOSA, 2013; BITTENCOURT; SAKELLARIOU; MADEIRA, 2010; ZHOU et al., 2016; ALEBRAHIM; AHMAD, 2016; SIH; LEE, 1993).
- **Evidência prática sobre "lista + inserção + antevisão".** Demonstramos empiricamente que políticas de inserção e antevisão (HEFT-LA, PEFT, IPEFT) tendem a preservar melhor o caminho crítico do que abordagens puramente dinâmicas em cenários com comunicação relevante (BITTENCOURT; SAKELLARIOU; MADEIRA, 2010; ARABNEJAD; BARBOSA, 2013; ZHOU et al., 2016).
- **Caracterização do DLS.** Documentamos o DLS como "padrão-ouro de balance-

amento" e com queda quase linear do *makespan* ao aumentar PEs em cargas com amplo paralelismo, porém com piores tempos quando dependências/CCR dominam (SIH; LEE, 1993; HAGRAS; JANEČEK, 2003).

- **Relevância para LLMs heterogêneos.** Os cenários e resultados conectam a literatura clássica de escalonamento às demandas atuais de treinamento/serving de LLMs em ambientes heterogêneos e federados (YANG et al., 2024; YAN et al., 2024; YAN et al., 2025), reforçando que escolher heurísticas com inserção/antevisão é, na prática, uma decisão de alto impacto tanto em desempenho quanto em energia.

Em síntese, as evidências deste trabalho corroboram as previsões das referências originais: **lista + inserção + antevisão** é a combinação que melhor conserva o caminho crítico em ambientes heterogêneos; o **DLS** permanece uma alternativa robusta quando o objetivo principal é balanceamento/escala, ciente do potencial custo em *makespan*. Essas conclusões, aliadas a um protocolo de avaliação reprodutível, oferecem um guia pragmático para a orquestração de *workflows* de ML/LLMs sobre clusters modernos.

Referências

ALEBRAHIM, S.; AHMAD, I. Task scheduling for heterogeneous computing systems. **The Journal of Supercomputing**, 2016. First online; final version in 2017 (73(6):2313–2338). Citado 9 vezes nas páginas 20, 21, 29, 30, 33, 43, 44, 127 e 128.

ARABNEJAD, H.; BARBOSA, J. G. List scheduling algorithm for heterogeneous systems by an optimistic cost table. **IEEE transactions on parallel and distributed systems**, IEEE, v. 25, n. 3, p. 682–694, 2013. Citado 18 vezes nas páginas 18, 19, 20, 21, 26, 29, 30, 32, 33, 39, 40, 41, 42, 43, 46, 47, 127 e 128.

BASKIYAR, S.; DICKINSON, C. Scheduling directed a-cyclic task graphs on a bounded set of heterogeneous processors using task duplication. **Journal of Parallel and Distributed Computing**, Elsevier, v. 65, n. 8, p. 911–921, 2005. Citado na página 35.

BITTENCOURT, L. F.; SAKELLARIOU, R.; MADEIRA, E. R. Dag scheduling using a lookahead variant of the heterogeneous earliest finish time algorithm. In: IEEE. **2010 18th Euromicro conference on parallel, distributed and network-based processing**. [S.l.], 2010. p. 27–34. Citado 10 vezes nas páginas 20, 21, 29, 30, 32, 33, 38, 39, 127 e 128.

BRAGHETTO, K. R.; CORDEIRO, D. Introdução à modelagem e execução de workflows científicos. In: **Jornadas de Atualização em Informática**. Porto Alegre: SBC, 2014. Citado na página 17.

CANON, L.-C.; JEANNOT, E.; SAKELLARIOU, R.; ZHENG, W. Comparative evaluation of the robustness of dag scheduling heuristics. In: **Grid Computing: Achievements and Prospects**. [S.l.]: Springer, 2008. p. 73–84. Citado na página 31.

COSTA, B. C. S. Avaliação de algoritmos de escalonamento de aplicações paralelas em processadores heterogêneos. Universidade Federal de Uberlândia, 2022. Citado 3 vezes nas páginas 20, 22 e 33.

HAGRAS, T.; JANEČEK, J. Static vs. dynamic list-scheduling performance comparison. **Acta Polytechnica**, v. 43, n. 6, 2003. Citado 6 vezes nas páginas 20, 34, 35, 37, 128 e 129.

HENDERSON, P.; HU, J.; ROMOFF, J.; BRUNSKILL, E.; JURAFSKY, D.; PINEAU, J. Towards the systematic reporting of the energy and carbon footprints of machine learning. **Journal of Machine Learning Research**, v. 21, n. 248, p. 1–43, 2020. Disponível em: <http://jmlr.org/papers/v21/20-312.html>. Citado 3 vezes nas páginas 21, 127 e 128.

KHAN, M. A. Scheduling for heterogeneous systems using constrained critical paths. **Parallel Computing**, v. 38, p. 175–193, 2012. Citado na página 136.

KNIGHT, W. **These Startups Are Building Advanced AI Models Without Data Centers**. 2025. *WIRED Magazine*. Accessed: 23 Jul 2025. Disponível em: <https://www.wired.com/story/

these-startups-are-building-advanced-ai-models-over-the-internet-with-untapped-data/>. Citado na página 18.

LACOSTE, A.; LUCCIONI, A.; SCHMIDT, V.; DANDRES, T. Quantifying the carbon emissions of machine learning. **arXiv preprint arXiv:1910.09700**, 2019. Disponível em: <https://arxiv.org/abs/1910.09700>. Citado 3 vezes nas páginas 21, 127 e 128.

MACK, J.; ARDA, S. E.; OGRAS, U. Y.; AKOGLU, A. Performant, multi-objective scheduling of highly interleaved task graphs on heterogeneous system on chip devices. **IEEE Transactions on Parallel and Distributed Systems**, IEEE, v. 33, n. 9, p. 2148–2162, 2021. Citado na página 37.

MONTEIRO, M. C. **Scheduling Algorithms for Heterogeneous GPU Task Graphs**. 2025. <https://github.com/costadev00/scheduling-algorithms-for-llm-training-in-heterogeneous-gpu>. GitHub repository; MIT License; Bachelor Final Project (TCC), UFU/FACOM. Version `82b6fb2` (commit: 82b6fb238205313fdc549438a573c060ef8b5139). Disponível em: <https://github.com/costadev00/scheduling-algorithms-for-llm-training-in-heterogeneous-gpu>. Citado na página 23.

NIYOM, A.; SOPHATSATHIT, P.; LURSINSAP, C. A fast predictive algorithm with idle reduction for heterogeneous system scheduling. **Simulation Modelling Practice and Theory**, Elsevier, v. 63, p. 83–103, 2016. Citado 2 vezes nas páginas 39 e 136.

PATTERSON, D.; GONZALEZ, J.; LE, Q. V.; LIANG, C.; MUNGUIA, D.; ROTHCHILD, D.; SO, D.; TEXIER, M.; DEAN, J. Carbon emissions and large neural network training. **arXiv preprint arXiv:2104.10350**, 2021. Disponível em: <https://arxiv.org/abs/2104.10350>. Citado 3 vezes nas páginas 21, 127 e 128.

SANDOKJI, S.; EASSA, F. Communication and computation aware task scheduling framework toward exascale computing. **International Journal of Advanced Computer Science and Applications**, Science and Information (SAI) Organization Limited, v. 10, n. 7, p. 119–128, 2019. Citado 4 vezes nas páginas 35, 37, 40 e 137.

SCHWARTZ, R.; DODGE, J.; SMITH, N. A.; ETZIONI, O. Green AI. **Communications of the ACM**, v. 63, n. 12, p. 54–63, 2020. Disponível em: <https://dl.acm.org/doi/10.1145/3381831>. Citado 3 vezes nas páginas 21, 127 e 128.

SIH, G. C.; LEE, E. A. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. **IEEE Transactions on Parallel and Distributed Systems**, v. 4, n. 2, p. 175–187, 1993. Citado 10 vezes nas páginas 20, 21, 29, 30, 32, 33, 37, 127, 128 e 129.

STRUBELL, E.; GANESH, A.; MCCALLUM, A. Energy and policy considerations for deep learning in NLP. In: **Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)**. [s.n.], 2019. p. 3645–3650. Disponível em: <https://aclanthology.org/P19-1355/>. Citado 3 vezes nas páginas 21, 127 e 128.

TOPCUOGLU, H.; HARIRI, S.; WU, M.-Y. Performance-effective and low-complexity task scheduling for heterogeneous computing. **IEEE transactions on parallel and distributed systems**, IEEE, v. 13, n. 3, p. 260–274, 2002. Citado 18 vezes nas páginas 18, 19, 20, 21, 26, 29, 30, 32, 33, 35, 36, 37, 38, 41, 43, 46, 127 e 128.

YAN, R.; JIANG, Y.; NIE, X.; FU, F.; CUI, B.; YUAN, B. **HexiScale: Accommodating Large Language Model Training over Heterogeneous Environment**. 2025. V2, 13 Mar 2025. Citado 4 vezes nas páginas 17, 31, 127 e 129.

YAN, R.; JIANG, Y.; TAO, W.; NIE, X.; CUI, B.; YUAN, B. Flashflex: Accommodating large language model training over heterogeneous environment. **arXiv e-prints**, p. arXiv–2409, 2024. Citado 6 vezes nas páginas 17, 18, 43, 127, 129 e 137.

YANG, F.; PENG, S.; SUN, N.; WANG, F.; WANG, Y.; WU, F.; QIU, J.; PAN, A. Holmes: Towards distributed training across clusters with heterogeneous nic environment. In: **Proceedings of the 53rd International Conference on Parallel Processing**. [S.l.: s.n.], 2024. p. 514–523. Citado 2 vezes nas páginas 127 e 129.

ZHOU, N.; QI, D.; WANG, X.; ZHENG, Z.; LIN, W. A list scheduling algorithm for heterogeneous systems based on a critical node cost table and pessimistic cost table. **Concurrency and Computation: Practice and Experience**, v. 29, n. 5, p. e3944, 2016. Disponível em: <https://onlinelibrary.wiley.com/doi/10.1002/cpe.3944>. Citado 11 vezes nas páginas 20, 21, 26, 29, 30, 32, 33, 45, 46, 127 e 128.

Apêndices

APÊNDICE A – Reprodutibilidade do Projeto e Benchmark Unificado

Este apêndice descreve, de forma **enxuta**, como reproduzir os experimentos e comparar **seis** heurísticas de *list scheduling* sobre **um mesmo dataset**, garantindo comparabilidade e repetibilidade. O ponto de entrada único é o script `compare_same_dataset.py`, que padroniza leitura de dados, execução e cálculo de métricas.

Repositório e instalação

Repositório: <https://github.com/costadev00/scheduling-algorithms-for-llm-training-in-heterogeneo

```
git clone https://github.com/costadev00/scheduling-algorithms-for-llm-training-in-h
cd scheduling-algorithms-for-llm-training-in-heterogeneous-gpu
python -m pip install -r requirements.txt
```

(Opcional) Instale Graphviz para melhorar o *layout* do `dag.png`. Sem Graphviz, um *fallback* é usado automaticamente.

Benchmark unificado (um comando)

Compare quaisquer subconjuntos de algoritmos `DLS,HEFT,HEFT-LA,PEFT,IHEFT,IPEFT` sobre um **mesmo** conjunto de CSVs:

```
python compare_same_dataset.py ‘
  --dag   <caminho>/task_connectivity.csv ‘
  --exec  <caminho>/task_exe_time.csv ‘
  --bw    <caminho>/resource_BW.csv ‘
  --power <caminho>/task_power.csv ‘   # opcional
  --algos DLS,HEFT,HEFT-LA,PEFT,IHEFT,IPEFT ‘
  --out_dir outputs/run_001 ‘          # opcional
  --save_dag --plot_metrics            # opcionais
```

Entradas e saídas (resumo)

Entradas (CSV): (i) conectividade (tarefas×tarefas, pesos = volume de dados), (ii) tempos de execução (tarefas×processadores), (iii) banda (processadores×processadores) e (iv) potência (opcional).
Saídas: linhas `JSON` no console (uma por algoritmo) + artefatos opcionais em `-out_dir`: `gantt_<ALG>.png`, `metrics.png` e `dag.png`.

Garantias de reprodutibilidade

- **I/O unificado e métricas comuns**: diferenças observadas refletem a *lógica* de cada algoritmo, não o instrumento de medição.
- **Determinismo dado o mesmo dataset**: para os algoritmos com componentes estocásticos, sementes fixas são utilizadas no baseline.
- **Ambiente controlado**: dependências versionadas via `requirements.txt`; execução *cross-platform* (Windows/Linux/macOS).
- **Traço de execução**: as linhas `JSON` permitem agregação e auditoria (`tee` para `.jsonl` quando desejado).

Checklist rápido

1. Clonar o repositório e instalar dependências.
2. Selecionar o **mesmo** conjunto de CSVs (DAG/exec/bw/*power* opcional).
3. Executar `compare_same_dataset.py` com a lista de algoritmos desejada.
4. (Opcional) Salvar artefatos em `-out_dir` e redirecionar métricas para `.jsonl`.

Observação: Para rodadas grandes, omita `-save_dag`; para iterações rápidas, reduza `-algos`.

APÊNDICE B – Trabalhos correlatos não incluídos na avaliação

Este apêndice registra heurísticas e sistemas *relacionados ao escalonamento em ambientes heterogêneos* que conhecemos, mas que **não** foram incluídos na avaliação experimental deste trabalho. Nosso objetivo principal foi comparar, lado a lado, os quatro algoritmos selecionados (DLS, HEFT, HEFT-LA e PEFT); as abordagens abaixo são listadas como referência para trabalhos futuros.

B.1 CEFT – Constrained Earliest Finish Time (2012)

O CEFT identifica e percorre *caminhos críticos constritos*–definidos por custos médios de execução, pesos de transmissão e influência de predecessores–podando o DAG de *tentry* a *texit* para extrair, em laços, o caminho válido mais longo e enfileirar as tarefas prontas (round-robin entre subcaminhos). Mantém a lógica "EFT + prioridade por caminho crítico", mas torna explícitas as restrições de comunicação/topologia na prioridade e na seleção; sua complexidade é $O(n\,p\,(n + m + \deg_{in}))$ (com n tarefas, m arestas, p processadores). Listamos aqui como trabalho correlato à família HEFT/PEFT/Lookahead para contexto, não incluído na nossa avaliação. Ver (KHAN, 2012).

B.2 PATS – Predict and Arrange Task Scheduling (2016)

Heurística de *list scheduling* em duas fases: (i) seleção por *EFT* preditivo executada *nível a nível* no DAG e (ii) redução explícita de *idle slots* por inserção, visando minimizar lacunas e o tempo total. Mantém baixa complexidade e, em avaliações com grafos sintéticos e aplicações reais (p.ex., Montage, FFT, Gaussian Elimination), apresentou SLR/makespan e tempo de execução superiores aos de HEFT/PEFT. Ver (NIYOM; SOPHATSATHIT; LURSINSAP, 2016).

B.3 DVR-HEFT e CCATSF – variantes dinâmicas (2019)

Linha de trabalhos que estende o **HEFT** com ranqueamento dinâmico (DVR-HEFT) e um *framework* consciente de comunicação e computação (CCATSF), integrando monitoramento e despacho em tempo de execução sem elevar a complexidade assintótica do HEFT. Em avaliações com DAGs sintéticos e aplicações, reportam **reduções de ma-**

kespan e melhor **eficiência** frente a HEFT, PEFT e Lookahead, especialmente quando há muitos nós e custos de comunicação relevantes (SANDOKJI; EASSA, 2019).

B.4 HexiScale – treinamento de LLMs em GPUs heterogêneas (2025)

Embora não seja uma "lista" clássica, o sistema **HexiScale** aplica **particionamento hierárquico de grafos** e alocação assimétrica para treino de LLMs em clusters **heterogêneos**, mantendo uso de recursos comparável a ambientes homogêneos de ponta. É particularmente relevante para nosso contexto aplicado (LLMs). Ver (YAN et al., 2024).

B.4.0.0.1 Observação metodológica.

Os algoritmos são executados sob cenários parametrizados por: número de processadores $|P|$, número de tarefas $|V|$, custo computacional $w_i^p > 0$ (tempo de execução de n_i em p), custo de comunicação $c_{a,b}(n_m, n_i)$ para $a \neq b$ na aresta (n_m, n_i) e custo de banda $B_{a,b}$ (transferência entre processadores). A avaliação pós-execução emprega as métricas *makespan*, *load balancing* e *waiting time*, formalizadas no capítulo de 2; trabalhos correlatos podem ser reavaliados no mesmo arcabouço experimental em extensões futuras.

B.4.0.0.2 Roteiro para trabalhos futuros.

Como continuação direta deste TCC, sugerimos: (i) incorporar PATS e CEFT ao *pipeline* atual, mantendo a mesma suíte de DAGs e cenários; (ii) avaliar DVR-HEFT/CCATSF em regimes com forte custo de comunicação; e (iii) cruzar as heurísticas clássicas com o particionamento hierárquico de grafos ao estilo HexiScale para LLMs.

Anexos

ANEXO A – Manifesto da Release e Licenças

A.1 Licenciamento do repositório

O código-fonte deste projeto, disponibilizado em <https://github.com/costadev00/scheduling-algorithms-for-llm-training-in-heterogeneous-gpu>, está licenciado sob a **Licença MIT (MIT License)**. O texto integral encontra-se no arquivo `LICENSE` do repositório (<https://github.com/costadev00/scheduling-algorithms-for-llm-training-in-heterogeneous-gpu/blob/main/LICENSE>). Em linhas gerais, a Licença MIT permite o uso, cópia, modificação, fusão, publicação, distribuição, sublicenciamento e/ou venda do software, desde que mantidos o aviso de *copyright* e a nota de licença.