Relatório
Laboratório I

Alunos: Maria Edoarda Vallim Fonseca
Thales Athayde Santos

Exercício I:
Erros ocorridos:
- Teve muita cópia e cola de código porque os dois membros da dupla interpretaram que era para separar a lógica da tarefa mestre dos empregados, apesar da lógica de execução ser a mesma.
- Tivemos dificuldade em fazer o código rodar em várias máquinas de início.
- O programa não é muito otimizado.

Acertos: O programa faz o que foi pedido.

	Teste 1	Teste 2	Teste 3	Teste 4	Teste 5	Média
1 processo 1 computador	0.001390	0.070369	0.055283	0.050513	0.044807	0,0444724
16 processos, 1 computador	0.240888	0.169658	0.290352	0.529903	0.269747	0,3001096
16 processos 4 computadores	0.388524	0.210070	0.160086	0.202734	0.289409	0,2501646
64 processos 1 computador	1.902749	1.479161	1.466650	1.290215	1.598320	1,547419
64 processos 4 computadores	1.550439	1.508820	1.419537	1.740468	1.205656	1,484984

Provavelmente ocorrerá uma melhora de desempenho mais significativa se as máquinas forem fisicamente diferentes.

Exercício II:
Acertos: Em relação ao exercício 1, o programa executou mais rápido, o que é de se esperar visto que os processos não estão sendo bloqueados.

	Teste 1	Teste 2	Teste 3	Teste 4	Teste 5	Média
1 processo 1 computador	0.013674	0.020836	0.000632	0.000496	0.000099	0,0071474
16 processos, 1 computador	0.390093	0.180133	0.147217	0.300186	0.111529	0,2258316
16 processos 4 computadores	0.220009	0.266284	0.143310	0.147880	0.14671	0,1848396
64 processos 1 computador	1.129776	0.956991	1.184407	1.029115	1.019211	1,0639
64 processos 4 computadores	1.082020	1.330007	2.049839	1.507104	1.656413	1,5250766

Exercício III:

Erros: Por algum motivo que a dupla não entendeu, dar free no malloc usado no código sempre dava segmentation fault, mesmo quando a gente fazia só o nó mestre usar a função. O free está comentado.
Sem free, ele dá segmentation fault mais ou menos 50% das vezes.
SOLUÇÃO: A gente deixou de alocar o array dinamicamente.
Como a máquina virtual em que testamos tem muita pouca memória, ela não suporta vetores com mais de 64 posições (testamos sempre com vetores em tamanho de potências de 2).

Acertos: Entendemos um pouco melhor como o MPI funciona em questão de separação de tarefas, então não há muita repetição inadequada de código!
As funções MPI_Gather e MPI_Scatter foram utilizadas corretamente.

Cada máquina tem 2 slots.

Nos testes a seguir, todos utilizaram a quantidade máxima de slots possível:

	Teste 1	Teste 2	Teste 3	Teste 4	Teste 5	Média
1 processo 1 computador	0.060980	0.056678	0.060616	0.022645	0.051769	0,0505376
1 processo 4 computadores	0.057226	0.020322	0.052608	0.004702	0.031218	0,0332152
16 processos 1 computador	0.511639	0.263984	0.452034	0.550216	0.579932	0,471561
16 processos 4 computadores	0.501808	0.500300	0.430786	0.440220	0.583783	0,4913794
32 processos 1 computador	1.072403	1.241227	1.076972	1.240825	1.168966	1,1600786
32 processos 4 computadores	1.292915	1.060691	1.562611	1.140570	1.250010	1,2613594

Nos testes a seguir, todos os computadores, exceto o mestre, utilizaram metade dos slots disponíveis (um slot):

	Teste 1	Teste 2	Teste 3	Teste 4	Teste 5	Média
1 processo 4 computadores	0.042420	0.044303	0.032445	0.020488	0.011274	0,030186
16 processos 4 computadores	0.553334	0.446912	0.490533	0.521164	0.522494	0,5068874
32 processos 4 computadores	1.161093	1.330778	1.367484	1.090646	1.013808	1,1927618

Comentário geral:
A dupla utilizou uma solução em containers para simular 4 computadores diferentes e realizar os testes, por não termos conseguido testar no laboratório. Os containers tinham acesso à 2 processadores e 1GB de RAM.
Como todos os recursos não eram fisicamente distintos, existe uma possibilidade alta dos ganhos em desempenho serem maiores em máquinas fisicamente distintas, mesmo levando em consideração o overhead da comunicação física de rede.
A perda de desempenho comparando 2 slots com 1 nos dois últimos testes do exercício 3 pode ter sido ocasionada pela grande quantidade de trocas de contexto dos processos.