Universidade do Minho

Mestrado Integrado em Engenharia Informática

Paradigmas da Computação Paralela

2º Trabalho - *MPI*

Bucket-Sort

João Alves (A77070)
Filipe Silva (A77284)

4 de Janeiro de 2019

Conteúdo

Lista de Figuras

1 Análise Teórica do Algoritmo

O algoritmo estudado pelo grupo, assim como para a implementação com memória partilhada anteriormente, é o ***bucket-sort***. Este é um algoritmo de ordenação, que foi descrito como podendo ser dividido em quatro fases distintas, mas dependentes. Sendo estas as seguintes: criação de um vetor de *buckets*, inicialmente vazios, colocação de cada elemento do vetor original, no *bucket* correspondente, ordenação dos *buckets* não vazios, aplicando um algoritmo de ordenação definido, e, finalmente, colocação no vetor original. Sendo todas as fases dependentes, estas têm de ser então executadas ordenadamente.

A análise da complexidade permite concluir que as fases de inicialização, colocação e recolocação no vetor original executam em tempo linear. Já a fase de ordenação dos *buckets* está dependente do algoritmo de ordenação que é aplicado. Uma vez que o algoritmo escolhido foi o *merge sort*, a sua complexidade é dada por $N \log_2 (N/p)$. Conclui-se assim que a complexidade total do algoritmo é igual a:

$$3N + N \log_2 (N/p) = O(N + N \log_2 (N/p))$$

Onde p é o número de *buckets* utilizados e N é o número de elementos do vetor inicial.

2 Desenvolvimento da Implementação MPI

Para proceder ao desenho da implementação **MPI**, o grupo decidiu seguir-se pela metodologia de **Foster** [1], constituída por quatro fases. Sendo que, cada uma destas será abordada nas próximas secções, com o intuito de justificar a solução implementada.

2.1 Partição do Problema e dos Dados a Processar

O primeiro contacto do grupo com o problema, na solução anterior com memória partilhada, permitiu-lhes visualizar algumas oportunidades de paralelismo.

Sendo que todas as fases do problema são dependentes, não será possível definir uma decomposição funcional. Contudo, existem dados que podem ser processados em paralelo, nomeadamente nas fases de inicialização e ordenação dos *buckets*, uma vez que, depois destes serem preenchidos, podem ser ordenados de forma independente. Com isto, foi então possível reconhecer a aplicação de um mesmo algoritmo de ordenação, a um conjunto de dados previamente processado (*buckets*). Sendo esta uma das principais características de um padrão ***master-slave***, o grupo decidiu trabalhar sobre esta hipótese de implementação.

Posto isto, sendo que o ***master*** fica responsável pela alocação dos elementos do vetor original, nos *buckets* respetivos, e os ***slaves*** pela ordenação de cada *bucket*, a divisão do trabalho é feita pelo ***master*** e enviada aos ***slaves***, para, depois de executadas as tarefas, os resultados poderem ser recolhidos pelo ***master***.

Concluiu-se assim que, este padrão torna-se adequado, uma vez que não existem dependências entre as tarefas a serem executadas e os custos de passagem de dados, entre o *master* e os *slaves*, não seriam excessivamente elevados.

Porém, se a granularidade das tarefas for muito fina, ou se existir um número demasiado elevado de *slaves*, o processo *master* poderá tornar-se um *bottleneck*. Esse é o principal problema de um paradigma *master-slave*.

Por isso, o grupo decidiu estabelecer um teto para o número de *bucktes* a serem ordenados, que consiste na limitação do mesmo, ao número de *slaves*. Desta forma, tenta-se garantir que granularidade das tarefas seja maior e o número de *slaves* não seja demasiado elevado. Contudo, é preciso ter especial atenção à distribuição dos elementos pelos *buckets*, uma vez que, se esta for demasiado desbalanceada, terá um grande impacto negativo na eficiência do algoritmo.

2.2 Identificação da Comunicação Necessária

Seguindo ainda o modelo *master-slave*, a comunicação necessária envolve apenas o ***master*** e os ***slaves***, para o envio das tarefas e recolha dos resultados.

Posto isto, o grupo começou por implementar a comunicação entre *master* e *slaves*, usando as primitivas **MPI_Send** e **MPI_Recv**. Porém, um estudo mais aprofundado da documentação do **MPI**, permitiu concluir que as operações coletivas são bastante mais eficientes, do que uma implementação de *gather* ou *scatter*, recorrendo apenas às primitivas **MPI_Send** e **MPI_Recv** [2].

Visto isto, para tirar o melhor partido das operações coletivas, o grupo decidiu utilizar as primitivas **MPI_Scatter** para enviar os *buckets* para os processos respetivos, seguido pelo **MPI_Gatherv**, para recolher os resultados. A principal diferença entre **MPI_Gather** e **MPI_Gatherv**, encontra-se no facto da última permitir especificar um *displacement* relativo ao *array* retornado como resultado. A sua utilidade encontra-se no facto de os elementos não serem distribuídos uniformemente pelos *buckets*, no entanto, o *array* especificado como *displacement*, permite ao *master* reconhecer onde colocar o conjunto de dados, retornados por um dado processo, relativamente ao *buffer* recebido. Isto permite ao *master* recolher imediatamente os elementos ordenados por cada *slave*, para o vetor original.

Concluiu-se então que a comunicação é **local**, uma vez que cada tarefa comunica apenas com o ***master*** e este com as restantes tarefas, **estruturada** e **estática** e, por último, síncrona.

2.3 Aglomeração de Computação e de Comunicação

Com o objetivo de reduzir os custos de comunicação, para além de tirar proveito da eficiência das operações coletivas, todos os elementos do vetor original são primeiramente colocados no *bucket* respetivo e só depois são enviados, para proceder à ordenação.

Sendo que, se o vetor original for constituído por N elementos, cada *bucket* suportará também N elementos, contando com o pior caso, em que a distribuição é feita apenas por um *bucket*.

Contudo, a distribuição foi feita com vista em ser o mais balanceada possível, processando-se da seguinte forma: para um elemento aleatório do vetor original i, o seu *bucket* correspondente é dado por: $i \times \frac{P}{N}$, onde P é o número de *buckets* (*slaves*) e N é o número total de elementos do vetor original, assim como o valor máximo, existente ou não, no *array*.

No que diz respeito à ordenação de cada *bucket*, não existe qualquer tipo de comunicação, ou seja, não é necessário considerar qualquer tipo de aglomeração para a tarefa de ordenação.

Finalmente, a comunicação inerente no retorno dos resultados, consiste apenas no envio dos respetivos vetores ordenados, por parte dos ***slaves***, para o **master**. Sendo que, cada um envia o número de elementos que ordenou, em vez do tamanho do *bucket* em si.

2.4 Mapeamento das Tarefas no Sistema

Seguindo ainda o mesmo padrão de implementação, o grupo decidiu distribuir os *slaves* pelos processadores e, consequentemente, distribuir a carga através de uma atribuição estática.

Deste modo, a cada *slave* (processador) seria atribuído um *bucket* para ser ordenado.

Mais à frente, serão estudadas várias técnicas de mapeamento, com o objetivo de encontrar a estratégia mais eficiente (ver anexo A).

3 Análise Teórica da Implementação Paralela

3.1 Análise da Computação e Comunicação

Recordando a análise da complexidade da implementação sequencial, o seu custo era $N + N\log_2(N/p)$. Porém, com a introdução da paralelização ao nível da ordenação dos *buckets*, o seu custo é reduzido para $(\frac{N}{p})\log_2(\frac{N}{p})$, onde p é o número de processos. Ficando assim com um tempo de computação igual a:

$$N + \frac{N\log_2(N/p)}{p}$$

Posto isto, as operações de comunicação do programa resumem-se a dois *scatters*, um do número de elementos em cada *bucket* e outro dos elementos de cada *bucket*, pelos respetivos processos, e um *gather*.

Considerando que o custo do *scatter* de N elementos custa $T_s + NT_d$, onde T_s é o *overhead* de criação e o T_d é o custo do envio de um elemento, o tempo total de comunicação é dado por:

$$3pT_s + pT_d + 2NT_d$$

3.2 Análise do *SpeedUp*

Vistos os custos computacionais e de comunicação, o tempo total da execução paralela é dado pela soma dos dois, da seguinte forma:

$$\begin{aligned} T_e &= N + \frac{N\log_2(N/p)}{p} + 3pT_s + pT_d + 2NT_d \\ &= O(N + p + \frac{N\log_2(N/p)}{p}) \end{aligned} \tag{1}$$

Isto aplica-se, assumindo que a carga dos processos é distribuída equitativamente por todos. Podendo então o *speedup* ser calculado por:

$$speedup = \frac{3N + N\log_2(N/p)}{N + \frac{N\log_2(N/p)}{p} + 3pT_s + pT_d + 2NT_d}$$

4 Experimentação

Feito o desenho e após a implementação da solução visualizada, o grupo procedeu à experimentação e medição de métricas, que permitissem analisar o desempenho do programa.

Para obter dados reproduzíveis em cada medição, foram executadas oito repetições para cada uma, calculando de seguida a mediana dos resultados obtidos. Para além disto, a cache foi sempre limpa antes de se proceder à execução do algoritmo.

Nas próximas secções, serão justificadas as decisões tomadas pelo grupo, nesta fase de experimentação.

4.1 Máquinas Utilizadas

Tal como foi sugerido, o grupo optou por usar sempre duas máquinas do *rack* **641** do *cluster* **SeARCH** [5], para correr todos os testes e recolha de resultados.

Este *rack* é constituído por um total de vinte nodos, cada um com dois **CPU**, ambos **Intel Xeon Processor E5-2650** [6], cuja arquitetura encontra-se documentada em anexo (ver anexo B).

Finalmente, para cada teste, foram também requisitados 32 cores lógicos, uma vez que este é o máximo suportado pelos CPU, tentando assim evitar a possível partilha de recursos com outros utilizadores.

4.2 Dados de Entrada

Uma vez que a distribuição da carga de cada *bucket* é um fator determinante, no que diz respeito ao impacto no desempenho do algoritmo, o grupo considerou que todos os diferentes tamanhos para o vetor inicial deveriam ser potências de dois. Isto tendo em conta que, o número de *buckets* seja também uma potência de dois, de modo a simplificar a divisão e tentar melhorar a distribuição dos elementos.

Com isto, o grupo decidiu testar a execução do algoritmo com um número de *buckets* variável, para um mesmo tamanho. Ou seja, para vetores com tamanhos 2048, 16384, 1048576 e 2048k foram recolhidas medições para 2, 4, 8, 16 e 32 *buckets*.

A dimensão dos vetores foi calculada com base nos tamanhos das *caches*, de maneira a que estes fossem completamente carregados para os vários níveis da hierarquia. Tendo assim o vetor com 2048 elementos completamente carregado na *cache* L1, 16384 na *cache* L2 e assim sucessivamente, tendo em conta que o vetor de maior dimensão ultrapassaria a capacidade da *cache* L3.

5 Análise do Perfil de Execução

Com o intuito de retirar e analisar o perfil de execução do programa e após uma pesquisa sobre as ferramentas disponíveis, que pudessem auxiliar o grupo nesta tarefa, optou-se por usar uma ferramenta designada por **Score-P** (*Scalable Performance Measurement Infrastructure for Parallel Codes*) [3]. Esta disponibiliza uma infraestrutura que permite medir o perfil de execução de uma aplicação, orientada para computação de elevado desempenho.

Após um estudo prévio da documentação [4] desta, procedeu-se à sua aplicação, para analisar o perfil de execução de um vetor inicial com 2048k elementos, obtendo o resultado que pode ser visto em anexo (ver anexo C).

A primeira linha dos resultados obtidos fornece uma estimativa sobre o tamanho total requerido, com a agregação de todos os processos. É uma informação útil para estimar o espaço requerido no disco. Neste caso sendo 141MB.

A segunda linha permite estimar o espaço de memória necessário por cada processo. No entanto, a memória reservada pela ferramenta em cada processo, no início da execução da aplicação, deve ser suficiente para que os dados se encontrem todos em memória, de maneira a evitar *flushes*, uma vez que estes podem causar algum ruído nas medições.

Para além disto, o **Score-P** requer ainda alguma memória adicional para manter as suas estruturas de dados, daí também ser disponibilizada uma estimativa da memória necessária em cada processo, na terceira linha do *output*.

As cinco linhas seguintes fornecem já alguma informação sobre o tempo de execução da aplicação, distribuído por certos grupos de funções. A coluna *time(s)* apresenta o tempo despendido em regiões pertencentes ao respetivo grupo, em segundos. Enquanto que a coluna seguinte possui a fração do tempo total, consumido por esse grupo. De seguida, a coluna *time/visit(us)* mostra o tempo médio por visita em micro-segundos. Por fim, a coluna *visits* indica número de vezes que essa região é invocada.

Para este algoritmo foram reconhecidos os grupos **ALL**, **USR**, **MPI** e **COM**. Sendo os mais relevantes os grupos **MPI** e **USR**, que contêm as funções **MPI** e as funções definidas pelo grupo, respetivamente.

Finalmente, as linhas seguintes mostram as informações vistas anteriormente, mas desta vez para cada região da aplicação, onde cada região é uma função chamada pelo programa.

Seguindo então com a análise dos resultados obtidos, é possível verificar que as operações que possuem mais impacto no desempenho do programa são **MPI_Init**, **MPI_Scatter**, **merge** e **sort**, como seria de esperar. É de notar que a função **MPI_Scatter** é chamada oito vezes, sendo que tem-se apenas quatro processos em execução. Isto acontece porque é necessário enviar para todos os processos, não só os *buckets* para serem ordenados, mas também o número de elementos contidos em cada um. De seguida, as operações **merge** e **sort** dizem respeito à aplicação do algoritmo *merge sort*, para ordenar os elementos de cada *bucket*.

6 Análise de Resultados

Uma vez traçado o perfil de execução da aplicação, procedeu-se então com a análise de resultados, para as medições realizadas. Estas consistem na medição do tempo total

de execução, somente os tempos de comunicação e no número de elementos ordenados por cada processo. Com estas medições o grupo pretende calcular a escalabilidade do programa, os custos de comunicação e o grau de balanceamento da carga pelos processos.

Cada uma destas métricas será discutida aprofundadamente nas próximas secções, juntamente com os resultados correspondentes obtidos.

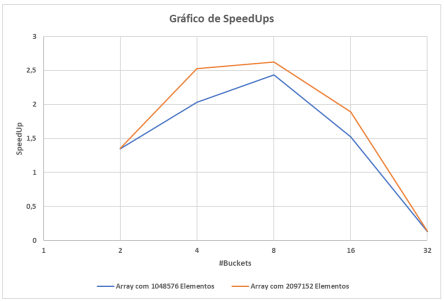
6.1 Escalabilidade

A primeira métrica analisada pelo grupo foi a escalabilidade do programa. Para tal, foi medido o tempo de execução total e, para as mesmas condições experimentais e dados de entrada, o tempo do algoritmo sequencial, dividindo o mesmo pelo resultado obtido da implementação paralela, obtendo por fim o ***speedup***.

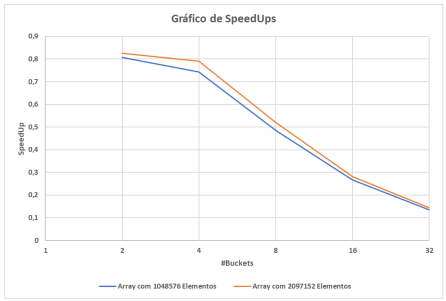
Uma primeira análise dos resultados obtidos, permite concluir que o algoritmo paralelo não possui ganhos para os dois vetores de menor dimensão. Uma vez que, o *overhead* associado à comunicação torna-se demasiado expressivo, face à granularidade das tarefas a serem executadas em paralelo. Na próxima secção verificar-se-á isto mesmo, comparando os tempos de comunicação e de computação.

De seguida, reparou-se que a aplicação escala de forma crescente até aos 8 processos por *core*, começando a decrescer a partir daí, devido à mesma razão discutida em cima (ver figura 1a). Contudo, a maior quebra ocorre, quando a distribuição é feita por 32 processos, causada pela utilização de duas máquinas diferentes, que aumenta em grande escala os tempos de comunicação.

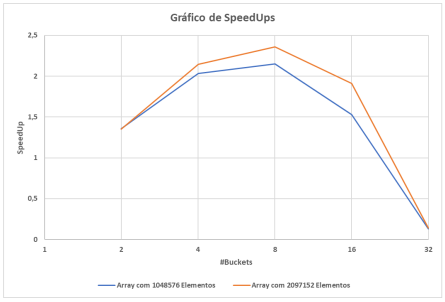
Para além disto, foram também recolhidos resultados para mais dois tipos diferentes de mapeamento dos processos, sendo estes por nodo e por *socket* (ver anexo D). Contudo, foi possível verificar que os melhores foram obtidos para os mapeamentos por *core* e por *socket* (ver anexo E), estando estes muito próximos um do outro. Para o mapeamento por nodo, como este é feito em *round-robin* por nodo, os processos serão atribuídos a *cores* em máquinas diferentes, o que implica elevados custos de comunicação entre eles.



(a) Mapeamento por *core*.



(b) Mapeamento por nodo.



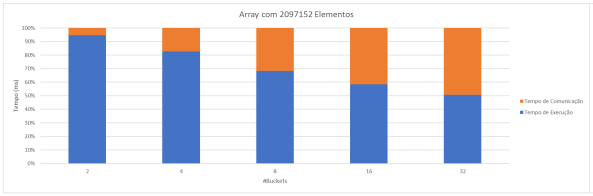
(c) Mapeamento por *socket*.

Figura 1: *Speedups* obtidos para os diferentes tipos de mapeamento.

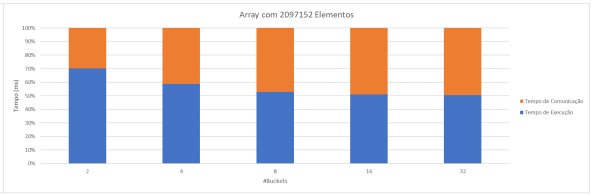
6.2 Custos de Comunicação

Para calcular os custos de comunicação, o grupo mediu apenas os tempos da distribuição dos *buckets* e respetivo número de elementos, pelos processos, e respetiva coleta resultados. Isto corresponde apenas às duas primitivas **MPI_Scatter** e à primitiva **MPI_Gatherv**.

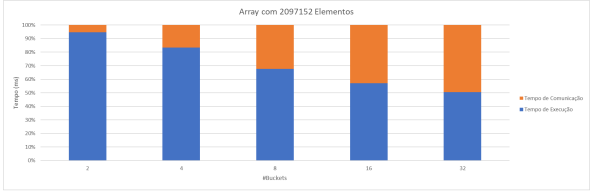
Obtidos os resultados, compararam-se os tempos de comunicação com os tempos de computação, que podem ser vistos na figura abaixo, para os respetivos tipos de mapeamento, e para o vetor de maior dimensão (2048k elementos).



(a) Mapeamento por *core*.



(b) Mapeamento por nodo.



(c) Mapeamento por *socket*.

Figura 2: Tempos de comunicação obtidos para os diferentes tipos de mapeamento, comparativamente com os tempos computacionais.

Como é possível concluir, os tempos de comunicação aumentam consideravelmente, a partir do momento em que são utilizadas duas máquinas.

6.3 Distribuição da Carga pelos Processos

Finalmente, para medir o grau de balanceamento ente a carga dos processos, foi contabilizado o número de elementos ordenado por cada processo, isto é o número de elementos por *bucket*. De seguida, foi feita a diferença entre o número máximo de elementos ordenados por um processo e o número mínimo, e foi calculada a percentagem de excesso (ver anexo G).

Com os resultados obtidos, foi possível reparar que quanto maior for o número de elementos, melhor é a distribuição dos mesmos, pelos *buckets* respetivos, melhorando também o desempenho do algoritmo.

A figura abaixo ilustra esta diferença, em percentagem, para o *array* de maior dimensão.

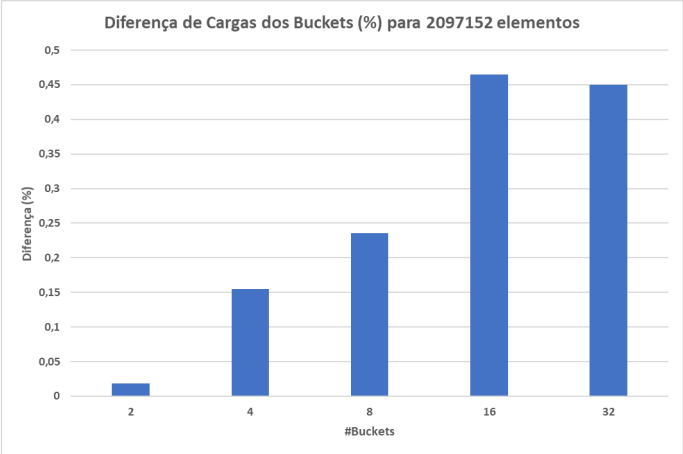


Figura 3: Diferença da carga distribuída pelos processos.

7 Comparação com a Implementação OpenMP e Conclusões

Uma vez que a implementação usando memória partilhada, com **OpenMP**, não exige comunicação entre *threads*, a sua escalabilidade será melhor, uma vez que não existe o *overhead* de comunicação.

Resta ainda concluir que, por causa do teto estabelecido para o número de *buckets*, a escalabilidade do programa encontra-se bastante limitada, uma vez que, se o número de *buckets* fosse superior e o mapeamento fosse dinâmico, talvez a escalabilidade do algoritmo melhorasse. Porém, isso exigiria que cada processo possuísse conhecimento sobre o número de *buckets* que já tinham sido ordenados, sendo que essa seria uma possível implementação híbrida (com memória partilhada e distribuída).

Referências

[1] Ian Foster. *Designing and Building Parallel Programs*. Addison-Wesley, 1995.

[2] Wes Kendall. *MPI Scatter, Gather, and Allgather*. URL: http://mpitutorial.com/tutorials/mpi-scatter-gather-and-allgather/.

[3] *Score-P*. URL: https://www.vi-hps.org/projects/score-p/.

[4] *SCORE-P User Manual*. URL: https://www.vi-hps.org/upload/packages/scorep/scorep-4.0.pdf.

[5] *SeARCH*. URL: http://search6.di.uminho.pt/wordpress/.

[6] *Search Nodes*. URL: http://search6.di.uminho.pt/wordpress/?page_id=55.

Anexos

A Mapeamento dos Processos

A.1 Por Core

Distribuição dos Processos com 2 Processos
MCW rank 0 bound to socket 0[core 0[hwt 0-1]]: [BB/../../../../../../../..][../../../../../../../../..]
MCW rank 1 bound to socket 0[core 1[hwt 0-1]]: [../BB/../../../../../../../..][../../../../../../../../..]

Distribuição dos Processos com 4 Processos
MCW rank 0 bound to socket 0[core 0[hwt 0-1]]: [BB/../../../../../../../..][../../../../../../../../..]
MCW rank 1 bound to socket 0[core 1[hwt 0-1]]: [../BB/../../../../../../../..][../../../../../../../../..]
MCW rank 2 bound to socket 0[core 2[hwt 0-1]]: [../../BB/../../../../../../..][../../../../../../../../..]
MCW rank 3 bound to socket 0[core 3[hwt 0-1]]: [../../../BB/../../../../../..][../../../../../../../../..]

Distribuição dos Processos com 8 Processos
MCW rank 0 bound to socket 0[core 0[hwt 0-1]]: [BB/../../../../../../../..][../../../../../../../..]
MCW rank 1 bound to socket 0[core 1[hwt 0-1]]: [../BB/../../../../../../..][../../../../../../../..]
MCW rank 2 bound to socket 0[core 2[hwt 0-1]]: [../../../BB/../../../../..][../../../../../../../..]
MCW rank 3 bound to socket 0[core 3[hwt 0-1]]: [../../../../BB/../../../..][../../../../../../../..]
MCW rank 4 bound to socket 0[core 4[hwt 0-1]]: [../../../../../BB/../../..][../../../../../../../..]
MCW rank 5 bound to socket 0[core 5[hwt 0-1]]: [../../../../../../BB/../..][../../../../../../../..]
MCW rank 6 bound to socket 0[core 6[hwt 0-1]]: [../../../../../../../BB/..][../../../../../../../..]
MCW rank 7 bound to socket 0[core 7[hwt 0-1]]: [../../../../../../../..][BB/../../../../../../../..]

Distribuição dos Processos com 16 Processos
MCW rank 0 bound to socket 0[core 0[hwt 0-1]]: [BB/../../../../../../../..][../../../../../../../..]
MCW rank 1 bound to socket 0[core 1[hwt 0-1]]: [../BB/../../../../../../..][../../../../../../../..]
MCW rank 2 bound to socket 0[core 2[hwt 0-1]]: [../../../BB/../../../../..][../../../../../../../..]
MCW rank 3 bound to socket 0[core 3[hwt 0-1]]: [../../../../BB/../../../..][../../../../../../../..]
MCW rank 4 bound to socket 0[core 4[hwt 0-1]]: [../../../../../BB/../../..][../../../../../../../..]
MCW rank 5 bound to socket 0[core 5[hwt 0-1]]: [../../../../../../BB/../..][../../../../../../../..]
MCW rank 6 bound to socket 0[core 6[hwt 0-1]]: [../../../../../../../BB/..][../../../../../../../..]
MCW rank 7 bound to socket 0[core 7[hwt 0-1]]: [../../../../../../../..][BB/../../../../../../../..]
MCW rank 8 bound to socket 1[core 8[hwt 0-1]]: [../../../../../../../..][BB/../../../../../../../..]
MCW rank 9 bound to socket 1[core 9[hwt 0-1]]: [../../../../../../../..][../BB/../../../../../../..]
MCW rank 10 bound to socket 1[core 10[hwt 0-1]]: [../../../../../../../..][../../BB/../../../../../..]
MCW rank 11 bound to socket 1[core 11[hwt 0-1]]: [../../../../../../../..][../../../BB/../../../../..]
MCW rank 12 bound to socket 1[core 12[hwt 0-1]]: [../../../../../../../..][../../../../BB/../../../..]
MCW rank 13 bound to socket 1[core 13[hwt 0-1]]: [../../../../../../../..][../../../../../BB/../../..]
MCW rank 14 bound to socket 1[core 14[hwt 0-1]]: [../../../../../../../..][../../../../../../BB/../..]
MCW rank 15 bound to socket 1[core 15[hwt 0-1]]: [../../../../../../../..][../../../../../../../BB]

Distribuição dos Processos com 32 Processos
MCW rank 0 bound to socket 0[core 0[hwt 0-1]]: [BB/../../../../../../../..][../../../../../../../..]
MCW rank 1 bound to socket 0[core 1[hwt 0-1]]: [../BB/../../../../../../..][../../../../../../../..]
MCW rank 2 bound to socket 0[core 2[hwt 0-1]]: [../../../BB/../../../../..][../../../../../../../..]
MCW rank 3 bound to socket 0[core 3[hwt 0-1]]: [../../../../BB/../../../..][../../../../../../../..]
MCW rank 4 bound to socket 0[core 4[hwt 0-1]]: [../../../../../BB/../../..][../../../../../../../..]
MCW rank 5 bound to socket 0[core 5[hwt 0-1]]: [../../../../../../BB/../..][../../../../../../../..]
MCW rank 6 bound to socket 0[core 6[hwt 0-1]]: [../../../../../../../BB/..][../../../../../../../..]
MCW rank 7 bound to socket 0[core 7[hwt 0-1]]: [../../../../../../../..][BB/../../../../../../../..]
MCW rank 8 bound to socket 1[core 8[hwt 0-1]]: [../../../../../../../..][BB/../../../../../../../..]
MCW rank 9 bound to socket 1[core 9[hwt 0-1]]: [../../../../../../../..][../BB/../../../../../../..]
MCW rank 10 bound to socket 1[core 10[hwt 0-1]]: [../../../../../../../..][../../BB/../../../../../..]
MCW rank 11 bound to socket 1[core 11[hwt 0-1]]: [../../../../../../../..][../../../BB/../../../../..]
MCW rank 12 bound to socket 1[core 12[hwt 0-1]]: [../../../../../../../..][../../../../BB/../../../..]
MCW rank 13 bound to socket 1[core 13[hwt 0-1]]: [../../../../../../../..][../../../../../BB/../../..]
MCW rank 14 bound to socket 1[core 14[hwt 0-1]]: [../../../../../../../..][../../../../../../BB/../..]
MCW rank 15 bound to socket 1[core 15[hwt 0-1]]: [../../../../../../../..][../../../../../../../BB]
MCW rank 16 bound to socket 0[core 0[hwt 0-1]]: [BB/../../../../../../../..][../../../../../../../..]
MCW rank 17 bound to socket 0[core 1[hwt 0-1]]: [../BB/../../../../../../..][../../../../../../../..]
MCW rank 18 bound to socket 0[core 2[hwt 0-1]]: [../../BB/../../../../../..][../../../../../../../..]
MCW rank 19 bound to socket 0[core 3[hwt 0-1]]: [../../../BB/../../../../..][../../../../../../../..]
MCW rank 20 bound to socket 0[core 4[hwt 0-1]]: [../../../../BB/../../../..][../../../../../../../..]
MCW rank 21 bound to socket 0[core 5[hwt 0-1]]: [../../../../../BB/../../..][../../../../../../../..]
MCW rank 22 bound to socket 0[core 6[hwt 0-1]]: [../../../../../../BB/../..][../../../../../../../..]
MCW rank 23 bound to socket 0[core 7[hwt 0-1]]: [../../../../../../../BB/..][../../../../../../../..]
MCW rank 24 bound to socket 1[core 8[hwt 0-1]]: [../../../../../../../..][BB/../../../../../../../..]
MCW rank 25 bound to socket 1[core 9[hwt 0-1]]: [../../../../../../../..][../BB/../../../../../../..]
MCW rank 26 bound to socket 1[core 10[hwt 0-1]]: [../../../../../../../..][../../BB/../../../../../..]
MCW rank 27 bound to socket 1[core 11[hwt 0-1]]: [../../../../../../../..][../../../BB/../../../../..]
MCW rank 28 bound to socket 1[core 12[hwt 0-1]]: [../../../../../../../..][../../../../BB/../../../..]
MCW rank 29 bound to socket 1[core 13[hwt 0-1]]: [../../../../../../../..][../../../../../BB/../../..]
MCW rank 30 bound to socket 1[core 14[hwt 0-1]]: [../../../../../../../..][../../../../../../BB/../..]
MCW rank 31 bound to socket 1[core 15[hwt 0-1]]: [../../../../../../../..][../../../../../../../BB]

A.2 Por Nodo

Distribuição dos Processos com 2 Processos
MCW rank 0 bound to socket 0[core 0[hwt 0-1]]: [BB/../../../../../../../..][../../../../../../../../..]
MCW rank 1 bound to socket 0[core 0[hwt 0-1]]: [BB/../../../../../../../..][../../../../../../../../..]

Distribuição dos Processos com 4 Processos
MCW rank 0 bound to socket 0[core 0[hwt 0-1]]: [BB/../../../../../../../..][../../../../../../../../..]
MCW rank 1 bound to socket 0[core 0[hwt 0-1]]: [BB/../../../../../../../..][../../../../../../../../..]
MCW rank 2 bound to socket 0[core 1[hwt 0-1]]: [../BB/../../../../../../..][../../../../../../../../..]
MCW rank 3 bound to socket 0[core 1[hwt 0-1]]: [../BB/../../../../../../..][../../../../../../../../..]

Distribuição dos Processos com 8 Processos
MCW rank 0 bound to socket 0[core 0[hwt 0-1]]: [BB/../../../../../../../..][../../../../../../../..]
MCW rank 1 bound to socket 0[core 0[hwt 0-1]]: [BB/../../../../../../../..][../../../../../../../..]
MCW rank 2 bound to socket 0[core 1[hwt 0-1]]: [../BB/../../../../../../..][../../../../../../../..]
MCW rank 3 bound to socket 0[core 1[hwt 0-1]]: [../BB/../../../../../../..][../../../../../../../..]
MCW rank 5 bound to socket 0[core 2[hwt 0-1]]: [../../BB/../../../../../..][../../../../../../../..]
MCW rank 4 bound to socket 0[core 2[hwt 0-1]]: [../../BB/../../../../../..][../../../../../../../..]
MCW rank 7 bound to socket 0[core 3[hwt 0-1]]: [../../../BB/../../../../..][../../../../../../../..]
MCW rank 6 bound to socket 0[core 3[hwt 0-1]]: [../../../BB/../../../../..][../../../../../../../..]

Distribuição dos Processos com 16 Processos
MCW rank 0 bound to socket 0[core 0[hwt 0-1]]: [BB/../../../../../../../..][../../../../../../../..]
MCW rank 1 bound to socket 0[core 0[hwt 0-1]]: [BB/../../../../../../../..][../../../../../../../..]
MCW rank 2 bound to socket 0[core 1[hwt 0-1]]: [../BB/../../../../../../..][../../../../../../../..]
MCW rank 3 bound to socket 0[core 1[hwt 0-1]]: [../BB/../../../../../../..][../../../../../../../..]
MCW rank 4 bound to socket 0[core 2[hwt 0-1]]: [../../BB/../../../../../..][../../../../../../../..]
MCW rank 5 bound to socket 0[core 2[hwt 0-1]]: [../../BB/../../../../../..][../../../../../../../..]
MCW rank 6 bound to socket 0[core 3[hwt 0-1]]: [../../../BB/../../../../..][../../../../../../../..]
MCW rank 7 bound to socket 0[core 3[hwt 0-1]]: [../../../BB/../../../../..][../../../../../../../..]
MCW rank 8 bound to socket 0[core 4[hwt 0-1]]: [../../../../BB/../../../..][../../../../../../../..]
MCW rank 9 bound to socket 0[core 4[hwt 0-1]]: [../../../../BB/../../../..][../../../../../../../..]
MCW rank 10 bound to socket 0[core 5[hwt 0-1]]: [../../../../../BB/../..][../../../../../../../..]
MCW rank 11 bound to socket 0[core 5[hwt 0-1]]: [../../../../../BB/../..][../../../../../../../..]
MCW rank 12 bound to socket 0[core 6[hwt 0-1]]: [../../../../../../BB/..][../../../../../../../..]
MCW rank 13 bound to socket 0[core 6[hwt 0-1]]: [../../../../../../BB/..][../../../../../../../..]
MCW rank 14 bound to socket 0[core 7[hwt 0-1]]: [../../../../../../../BB][../../../../../../../..]
MCW rank 15 bound to socket 0[core 7[hwt 0-1]]: [../../../../../../../BB][../../../../../../../..]

Distribuição dos Processos com 32 Processos
MCW rank 0 bound to socket 0[core 0[hwt 0-1]]: [BB/../../../../../../../..][../../../../../../../..]
MCW rank 1 bound to socket 0[core 0[hwt 0-1]]: [BB/../../../../../../../..][../../../../../../../..]
MCW rank 2 bound to socket 0[core 1[hwt 0-1]]: [../BB/../../../../../../..][../../../../../../../..]
MCW rank 3 bound to socket 0[core 1[hwt 0-1]]: [../BB/../../../../../../..][../../../../../../../..]
MCW rank 4 bound to socket 0[core 2[hwt 0-1]]: [../../BB/../../../../../..][../../../../../../../..]
MCW rank 5 bound to socket 0[core 2[hwt 0-1]]: [../../BB/../../../../../..][../../../../../../../..]
MCW rank 6 bound to socket 0[core 3[hwt 0-1]]: [../../../BB/../../../../..][../../../../../../../..]
MCW rank 7 bound to socket 0[core 3[hwt 0-1]]: [../../../BB/../../../../..][../../../../../../../..]
MCW rank 8 bound to socket 0[core 4[hwt 0-1]]: [../../../../BB/../../../..][../../../../../../../..]
MCW rank 9 bound to socket 0[core 4[hwt 0-1]]: [../../../../BB/../../../..][../../../../../../../..]
MCW rank 10 bound to socket 0[core 5[hwt 0-1]]: [../../../../../BB/../..][../../../../../../../..]
MCW rank 11 bound to socket 0[core 5[hwt 0-1]]: [../../../../../BB/../..][../../../../../../../..]
MCW rank 12 bound to socket 0[core 6[hwt 0-1]]: [../../../../../../BB/..][../../../../../../../..]
MCW rank 13 bound to socket 0[core 6[hwt 0-1]]: [../../../../../../BB/..][../../../../../../../..]
MCW rank 14 bound to socket 0[core 7[hwt 0-1]]: [../../../../../../../BB][../../../../../../../..]
MCW rank 15 bound to socket 0[core 7[hwt 0-1]]: [../../../../../../../BB][../../../../../../../..]
MCW rank 16 bound to socket 1[core 8[hwt 0-1]]: [../../../../../../../..][BB/../../../../../../..]
MCW rank 17 bound to socket 1[core 8[hwt 0-1]]: [../../../../../../../..][BB/../../../../../../..]
MCW rank 18 bound to socket 1[core 9[hwt 0-1]]: [../../../../../../../..][../BB/../../../../../..]
MCW rank 19 bound to socket 1[core 9[hwt 0-1]]: [../../../../../../../..][../BB/../../../../../..]
MCW rank 20 bound to socket 1[core 10[hwt 0-1]]: [../../../../../../../..][../../BB/../../../../..]
MCW rank 21 bound to socket 1[core 10[hwt 0-1]]: [../../../../../../../..][../../BB/../../../../..]
MCW rank 22 bound to socket 1[core 11[hwt 0-1]]: [../../../../../../../..][../../../BB/../../../..]
MCW rank 23 bound to socket 1[core 11[hwt 0-1]]: [../../../../../../../..][../../../BB/../../../..]
MCW rank 24 bound to socket 1[core 12[hwt 0-1]]: [../../../../../../../..][../../../../BB/../../..]
MCW rank 25 bound to socket 1[core 12[hwt 0-1]]: [../../../../../../../..][../../../../BB/../../..]
MCW rank 26 bound to socket 1[core 13[hwt 0-1]]: [../../../../../../../..][../../../../../BB/../..]
MCW rank 27 bound to socket 1[core 13[hwt 0-1]]: [../../../../../../../..][../../../../../BB/../..]
MCW rank 28 bound to socket 1[core 14[hwt 0-1]]: [../../../../../../../..][../../../../../../BB/..]
MCW rank 29 bound to socket 1[core 14[hwt 0-1]]: [../../../../../../../..][../../../../../../BB/..]
MCW rank 30 bound to socket 1[core 15[hwt 0-1]]: [../../../../../../../..][../../../../../../../BB]
MCW rank 31 bound to socket 1[core 15[hwt 0-1]]: [../../../../../../../..][../../../../../../../BB]

A.3 Por Socket

Distribuição dos Processos com 2 Processos
MCW rank 0 bound to socket 0[core 0[hwt 0-1]]: [BB/../../../../../../../..][../../../../../../../..]
MCW rank 1 bound to socket 1[core 8[hwt 0-1]]: [../../../../../../../..][BB/../../../../../../../..]

Distribuição dos Processos com 4 Processos
MCW rank 0 bound to socket 0[core 0[hwt 0-1]]: [BB/../../../../../../../..][../../../../../../../..]
MCW rank 1 bound to socket 1[core 8[hwt 0-1]]: [../../../../../../../..][BB/../../../../../../../..]
MCW rank 2 bound to socket 0[core 1[hwt 0-1]]: [../BB/../../../../../../..][../../../../../../../..]
MCW rank 3 bound to socket 1[core 9[hwt 0-1]]: [../../../../../../../..][../BB/../../../../../../..]

Distribuição dos Processos com 8 Processos
MCW rank 0 bound to socket 0[core 0[hwt 0-1]]: [BB/../../../../../../../..][../../../../../../../..]
MCW rank 1 bound to socket 1[core 8[hwt 0-1]]: [../../../../../../../..][BB/../../../../../../../..]
MCW rank 2 bound to socket 0[core 1[hwt 0-1]]: [../BB/../../../../../../..][../../../../../../../..]
MCW rank 3 bound to socket 1[core 9[hwt 0-1]]: [../../../../../../../..][../BB/../../../../../../..]
MCW rank 4 bound to socket 0[core 2[hwt 0-1]]: [../../BB/../../../../../..][../../../../../../../..]
MCW rank 5 bound to socket 1[core 10[hwt 0-1]]: [../../../../../../../..][../../BB/../../../../../..]
MCW rank 6 bound to socket 0[core 3[hwt 0-1]]: [../../../BB/../../../../..][../../../../../../../..]
MCW rank 7 bound to socket 1[core 11[hwt 0-1]]: [../../../../../../../..][../../../BB/../../../../..]

Distribuição dos Processos com 16 Processos
MCW rank 0 bound to socket 0[core 0[hwt 0-1]]: [BB/../../../../../../../..][../../../../../../../..]
MCW rank 1 bound to socket 1[core 8[hwt 0-1]]: [../../../../../../../..][BB/../../../../../../../..]
MCW rank 2 bound to socket 0[core 1[hwt 0-1]]: [../BB/../../../../../../..][../../../../../../../..]
MCW rank 3 bound to socket 1[core 9[hwt 0-1]]: [../../../../../../../..][../BB/../../../../../../..]
MCW rank 4 bound to socket 0[core 2[hwt 0-1]]: [../../BB/../../../../../..][../../../../../../../..]
MCW rank 5 bound to socket 1[core 10[hwt 0-1]]: [../../../../../../../..][../../BB/../../../../../..]
MCW rank 6 bound to socket 0[core 3[hwt 0-1]]: [../../../BB/../../../../..][../../../../../../../..]
MCW rank 7 bound to socket 1[core 11[hwt 0-1]]: [../../../../../../../..][../../../BB/../../../../..]
MCW rank 8 bound to socket 0[core 4[hwt 0-1]]: [../../../../BB/../../../..][../../../../../../../..]
MCW rank 9 bound to socket 1[core 12[hwt 0-1]]: [../../../../../../../..][../../../../BB/../../../..]
MCW rank 10 bound to socket 0[core 5[hwt 0-1]]: [../../../../../BB/../..][../../../../../../../..]
MCW rank 11 bound to socket 1[core 13[hwt 0-1]]: [../../../../../../../..][../../../../../BB/../..]
MCW rank 12 bound to socket 0[core 6[hwt 0-1]]: [../../../../../../BB/..][../../../../../../../..]
MCW rank 13 bound to socket 1[core 14[hwt 0-1]]: [../../../../../../../..][../../../../../../BB/..]
MCW rank 14 bound to socket 0[core 7[hwt 0-1]]: [../../../../../../../BB][../../../../../../../..]
MCW rank 15 bound to socket 1[core 15[hwt 0-1]]: [../../../../../../../..][../../../../../../../BB]

Distribuição dos Processos com 32 Processos
MCW rank 0 bound to socket 0[core 0[hwt 0-1]]: [BB/../../../../../../../..][../../../../../../../..]
MCW rank 1 bound to socket 1[core 8[hwt 0-1]]: [../../../../../../../..][BB/../../../../../../../..]
MCW rank 2 bound to socket 0[core 1[hwt 0-1]]: [../BB/../../../../../../..][../../../../../../../..]
MCW rank 3 bound to socket 1[core 9[hwt 0-1]]: [../../../../../../../..][../BB/../../../../../../..]
MCW rank 4 bound to socket 0[core 2[hwt 0-1]]: [../../BB/../../../../../..][../../../../../../../..]
MCW rank 5 bound to socket 1[core 10[hwt 0-1]]: [../../../../../../../..][../../BB/../../../../../..]
MCW rank 6 bound to socket 0[core 3[hwt 0-1]]: [../../../BB/../../../../..][../../../../../../../..]
MCW rank 7 bound to socket 1[core 11[hwt 0-1]]: [../../../../../../../..][../../../BB/../../../../..]
MCW rank 8 bound to socket 0[core 4[hwt 0-1]]: [../../../../BB/../../../..][../../../../../../../..]
MCW rank 9 bound to socket 1[core 12[hwt 0-1]]: [../../../../../../../..][../../../../BB/../../../..]
MCW rank 10 bound to socket 0[core 5[hwt 0-1]]: [../../../../../BB/../..][../../../../../../../..]
MCW rank 11 bound to socket 1[core 13[hwt 0-1]]: [../../../../../../../..][../../../../../BB/../..]
MCW rank 12 bound to socket 0[core 6[hwt 0-1]]: [../../../../../../BB/..][../../../../../../../..]
MCW rank 13 bound to socket 1[core 14[hwt 0-1]]: [../../../../../../../..][../../../../../../BB/..]
MCW rank 14 bound to socket 0[core 7[hwt 0-1]]: [../../../../../../../BB][../../../../../../../..]
MCW rank 15 bound to socket 1[core 15[hwt 0-1]]: [../../../../../../../..][../../../../../../../BB]
MCW rank 16 bound to socket 0[core 0[hwt 0-1]]: [BB/../../../../../../../..][../../../../../../../..]
MCW rank 17 bound to socket 1[core 8[hwt 0-1]]: [../../../../../../../..][BB/../../../../../../../..]
MCW rank 18 bound to socket 0[core 1[hwt 0-1]]: [../BB/../../../../../../..][../../../../../../../..]
MCW rank 19 bound to socket 1[core 9[hwt 0-1]]: [../../../../../../../..][../BB/../../../../../../..]
MCW rank 20 bound to socket 0[core 2[hwt 0-1]]: [../../BB/../../../../../..][../../../../../../../..]
MCW rank 21 bound to socket 1[core 10[hwt 0-1]]: [../../../../../../../..][../../BB/../../../../../..]
MCW rank 22 bound to socket 0[core 3[hwt 0-1]]: [../../../BB/../../../../..][../../../../../../../..]
MCW rank 23 bound to socket 1[core 11[hwt 0-1]]: [../../../../../../../..][../../../BB/../../../../..]
MCW rank 24 bound to socket 0[core 4[hwt 0-1]]: [../../../../BB/../../../..][../../../../../../../..]
MCW rank 25 bound to socket 1[core 12[hwt 0-1]]: [../../../../../../../..][../../../../BB/../../../..]
MCW rank 26 bound to socket 0[core 5[hwt 0-1]]: [../../../../../BB/../..][../../../../../../../..]
MCW rank 27 bound to socket 1[core 13[hwt 0-1]]: [../../../../../../../..][../../../../../BB/../..]
MCW rank 28 bound to socket 0[core 6[hwt 0-1]]: [../../../../../../BB/..][../../../../../../../..]
MCW rank 29 bound to socket 1[core 14[hwt 0-1]]: [../../../../../../../..][../../../../../../BB/..]
MCW rank 30 bound to socket 0[core 7[hwt 0-1]]: [../../../../../../../BB][../../../../../../../..]
MCW rank 31 bound to socket 1[core 15[hwt 0-1]]: [../../../../../../../..][../../../../../../../BB]

B Especificações dos CPUs Instalados nas Máquinas Usadas

Intel® Xeon® Processor E5-2650 v2	
Code Name	Ivy Bridge EP
Essentials	
Vertical Segment	Server
Processor Number	E5-2650V2
Performance	
# of Cores	8
# of Threads	16
Processor Base Frequency	2.60 GHz
Max Turbo Frequency	3.40 GHz
Cache	20 MB SmartCache
Bus Speed	8 GT/s QPI
# of QPI Links	2
TDP	95 W
VID Voltage Range	0.65–1.30V
Memory Specifications	
Max Memory Size (dependent on memory type)	768 GB
Memory Types	DDR3 800/1066/1333/1600/1866
Max # of Memory Channels	4
Max Memory Bandwidth	59.7 GB/s
Physical Address Extensions	46-bit
ECC Memory Supported ‡	Yes
Expansion Options	
Scalability	2S Only
PCI Express Revision	3.0
PCI Express Configurations ‡	x4, x8, x16
Max # of PCI Express Lanes	40
Advanced Technologies	
Intel® Turbo Boost Technology	2.0
Intel® vPro™ Platform Eligibility ‡	Yes
Intel® Hyper-Threading Technology ‡	Yes
Intel® Virtualization Technology (VT-x) ‡	Yes
Intel® Virtualization Technology for Directed I/O (VT-d) ‡	Yes
Intel® VT-x with Extended Page Tables (EPT) ‡	Yes
Intel® TSX-NI	No
Intel® 64 ‡	Yes
Instruction Set	64-bit
Instruction Set Extensions	Intel® AVX
Idle States	Yes
Enhanced Intel SpeedStep® Technology	Yes
Intel® Demand Based Switching	Yes
Thermal Monitoring Technologies	Yes
Intel® Flex Memory Access	No
Intel® Identity Protection Technology ‡	No

C Resultados da Análise do Perfil de Execução

Análise do Perfil de Execução para vetor com 2097152 elementos e 4 Buckets						
Estimated aggregate size of event trace						141MB
Estimated requirements for largest trace buffer (max_buf)						36MB
Estimated memory requirements (SCOREP_TOTAL_MEMORY)						4097kB
type	max_buf[B]	visits	time[s]	time[%]	time/visit[us]	region
ALL	36,929,148	5,676,414	2,29	100	0,4	ALL
USR	36,928,814	5,676,382	0,92	40,2	0,16	USR
MPI	308	28	1,37	59,8	48933,79	MPI
COM	26	4	0	0	36,15	COM
USR	27,283,074	4,194,300	0,51	22,4	0,12	void sort(int*, int*, int, int)
USR	9,645,688	1,482,066	0,27	12	0,19	void merge(int*, int*, int, int, int)
MPI	136	8	0,44	19,3	55325,94	MPI_Scatter
MPI	68	4	0,01	0,5	3116,37	MPI_Gatherv
MPI	26	4	0	0,2	882,01	MPI_Finalize
USR	26	4	0	0	11,43	void utils_results()
MPI	26	4	0	0	0,22	MPI_Comm_size
USR	26	1	0	0	451,73	int utils_clean(int*)
USR	26	1	0	0	3,2	void utils_setup(char*, int)
USR	26	1	0	0	2,83	void utils_start_timer()
USR	26	1	0,05	2	46621,47	void utils_init(int**, int)
USR	26	1	0,08	3,4	78333,39	void utils_clear_cache()
MPI	26	4	0	0	1,31	MPI_Comm_rank
USR	26	1	0	0	0,7	void bucketsort_init(int*, int)
USR	26	4	0	0	142,4	void mergesort(int*, int)
USR	26	1	0	0	5,77	void utils_stop_timer()
MPI	26	4	0,91	39,8	227884,73	MPI_Init
COM	26	4	0	0	36,15	int main(int, char**)
USR	26	1	0,01	0,3	6627,44	void bucketsort_placement(int*, int*, int*, int, int)

D Resultados Obtidos

D.1 Por Core

mapping: --map-by core				
Array com 2048 Elementos				
#Buckets	Tempo de Execução (ms)	Tempo de Comunicação (ms)	Tempo de Computação (ms)	Diferença de Cargas dos Buckets (%)
2	0,282	0,158906	0,124978	10,6996
4	0,275	0,19598	0,07752	12,5514
8	0,4115	0,352859	0,061078	9,87654
16	0,565	0,511527	0,0515425	12,6374
32	6,014	5,896565	-0,01302	27,1523
Array com 16384 Elementos				
#Buckets	Tempo de Execução (ms)	Tempo de Comunicação (ms)	Tempo de Computação (ms)	Diferença de Cargas dos Buckets (%)
2	1,1985	0,2980235	0,8999995	1,27764
4	0,9575	0,4669425	0,489541	2,27892
8	1,1215	0,7914305	0,3290695	6,1245
16	1,664	1,47104	0,19648	2,82353
32	13,5635	13,12195	0,3485	4,51745
Array com 1048576 Elementos				
#Buckets	Tempo de Execução (ms)	Tempo de Comunicação (ms)	Tempo de Computação (ms)	Diferença de Cargas dos Buckets (%)
2	78,1705	4,728435	73,36297	0,000953674
4	48,6965	9,165885	38,97544	0,186909
8	38,5665	18,4034	20,27265	0,138832
16	59,3725	40,67505	17,86845	0,544784
32	658,682	639,318	19,897	0,748081
Array com 2097152 Elementos				
#Buckets	Tempo de Execução (ms)	Tempo de Comunicação (ms)	Tempo de Computação (ms)	Diferença de Cargas dos Buckets (%)
2	161,903	9,275915	152,617555	0,0181215
4	83,7435	17,4946	66,47745	0,155003
8	74,9525	34,8054	40,18195	0,235977
16	99,318	70,88705	28,8835	0,464488
32	1291,92	1262,885	29,055	0,450237

D.2 Por Nodo

mapping: --map-by node				
Array com 2048 Elementos				
#Buckets	Tempo de Execução (ms)	Tempo de Comunicação (ms)	Tempo de Computação (ms)	Diferença de Cargas dos Buckets (%)
2	0,8165	0,6594655	0,154574	10,6996
4	1,051	1,01447	0,0495235	12,5514
8	1,6755	1,571895	0,086995	9,87654
16	3,094	2,922535	0,18243	2,82353
32	5,071	5,260585	0,054925	27,1523
Array com 16384 Elementos				
#Buckets	Tempo de Execução (ms)	Tempo de Comunicação (ms)	Tempo de Computação (ms)	Diferença de Cargas dos Buckets (%)
2	3,105	2,3216	0,83036	1,27764
4	3,2325	2,66743	0,66795	2,27892
8	4,3875	3,887535	0,480045	6,1245
16	6,873	6,549475	0,386975	2,82353
32	12,551	12,094	0,5211	4,51745
Array com 1048576 Elementos				
#Buckets	Tempo de Execução (ms)	Tempo de Comunicação (ms)	Tempo de Computação (ms)	Diferença de Cargas dos Buckets (%)
2	130,681	57,26205	73,5149	0,000953674
4	133,2235	94,76695	37,9345	0,186909
8	193,4125	171,254	22,3255	0,138832
16	339,4215	322,418	17,473	0,544784
32	640,189	623,2435	14,4295	0,748081
Array com 2097152 Elementos				
#Buckets	Tempo de Execução (ms)	Tempo de Comunicação (ms)	Tempo de Computação (ms)	Diferença de Cargas dos Buckets (%)
2	264,881	112,7005	152,171	0,0181215
4	267,392	188,83	78,224	0,155003
8	379,1955	339,8675	38,7975	0,235977
16	669,8585	643,462	26,907	0,464488
32	1275,525	1246,91	28,225	0,450237

D.3 Por Socket

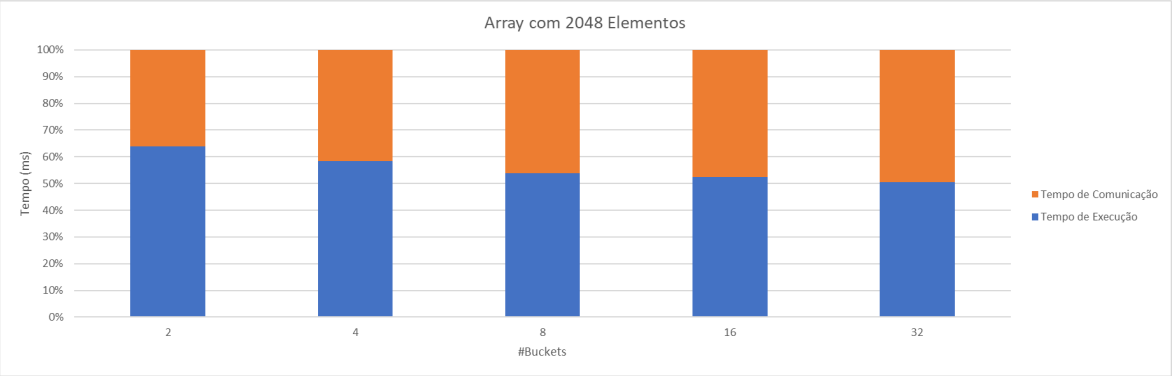
mapping: --map-by socket				
Array com 2048 Elementos				
#Buckets	Tempo de Execução (ms)	Tempo de Comunicação (ms)	Tempo de Computação (ms)	Diferença de Cargas dos Buckets (%)
2	0,2615	0,144005	0,1175415	10,6996
4	0,25	0,1859665	0,0690565	12,5514
8	0,3785	0,3111365	0,0688935	9,87654
16	0,6065	0,5345345	0,0759825	12,6374
32	5,7115	5,66888	0,113025	27,1523
Array com 16384 Elementos				
#Buckets	Tempo de Execução (ms)	Tempo de Comunicação (ms)	Tempo de Computação (ms)	Diferença de Cargas dos Buckets (%)
2	1,17	0,2815725	0,888497	1,27764
4	0,946	0,470996	0,475004	2,27892
8	1,13	0,8124115	0,3080155	6,1245
16	1,707	1,53196	0,17008	2,82353
32	13,595	13,29145	0,35955	4,51745
Array com 1048576 Elementos				
#Buckets	Tempo de Execução (ms)	Tempo de Comunicação (ms)	Tempo de Computação (ms)	Diferença de Cargas dos Buckets (%)
2	77,7465	4,40085	73,423055	0,000953674
4	48,6285	10,1106	38,5094	0,186909
8	43,676	20,96795	22,46445	0,138832
16	59,0945	40,8845	18,326	0,544784
32	658,451	639,8685	17,7565	0,748081
Array com 2097152 Elementos				
#Buckets	Tempo de Execução (ms)	Tempo de Comunicação (ms)	Tempo de Computação (ms)	Diferença de Cargas dos Buckets (%)
2	161,9865	9,317995	152,69654	0,0181215
4	98,5615	19,52505	79,30585	0,155003
8	83,5275	39,96755	43,2229	0,235977
16	98,396	73,79195	27,15045	0,464488
32	1291,375	1260,825	29,375	0,450237

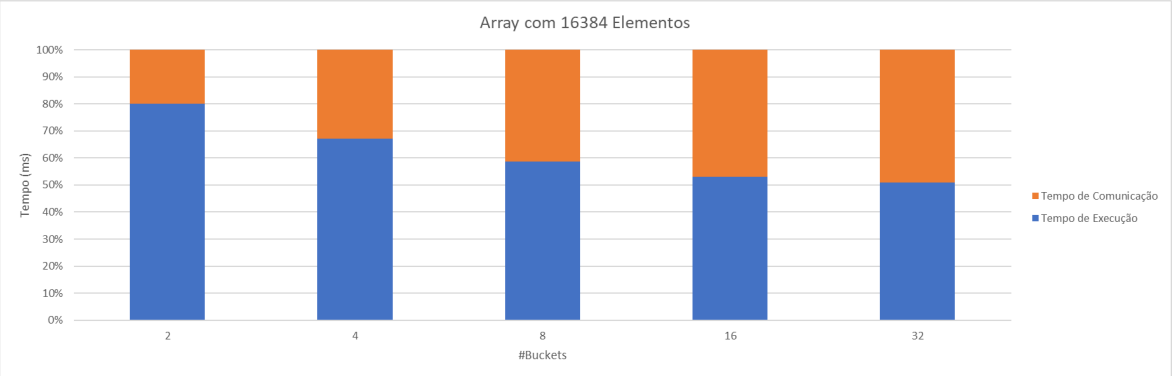
E Tabela de SpeedUps

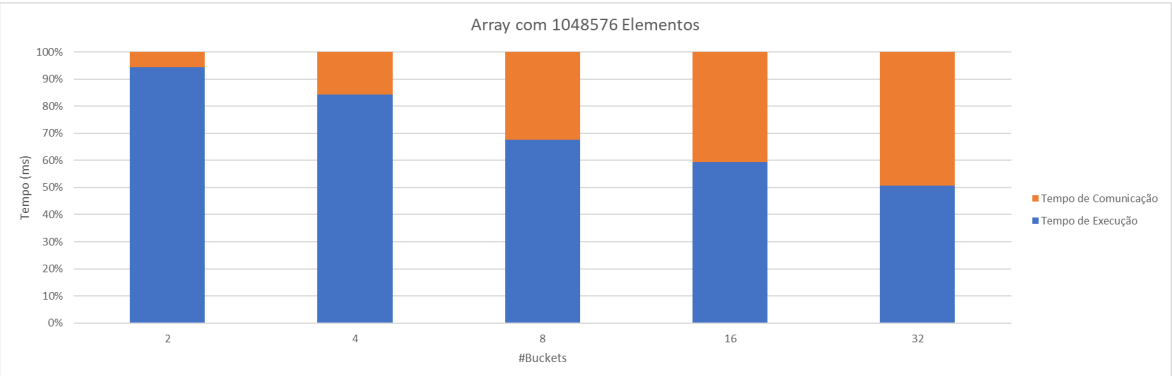
SpeedUps				
Array com 2048 Elementos				
#Buckets	OMP	MPI --map-by core	MPI --map-by node	MPI --map-by socket
2	0,616600791	0,553191489	0,1910594	0,596558317
4	0,640718563	0,389090909	0,101807802	0,428
8	0,646666667	0,235722965	0,057893166	0,256274769
16	0,55	0,175221239	0,031997414	0,163231657
32	0,408695652	0,015630196	0,018536778	0,016458023
Array com 16384 Elementos				
#Buckets	OMP	MPI --map-by core	MPI --map-by node	MPI --map-by socket
2	0,879339784	1,289111389	0,497584541	1,320512821
4	1,148546825	1,114360313	0,330085073	1,127906977
8	1,726027397	0,898796255	0,22974359	0,892035398
16	2,4921875	0,575120192	0,139240506	0,560632689
32	0,076842564	0,064953736	0,07019361	0,064803236
Array com 1048576 Elementos				
#Buckets	OMP	MPI --map-by core	MPI --map-by node	MPI --map-by socket
2	1,418197937	1,350662974	0,807936884	1,358028979
4	2,336472421	2,032877106	0,743067102	2,035719794
8	4,225073149	2,433718383	0,485284043	2,149006319
16	5,976344654	1,523365194	0,26647104	1,530531606
32	3,944581958	0,130647262	0,134421241	0,130693096
Array com 2097152 Elementos				
#Buckets	OMP	MPI --map-by-core	MPI --map-by node	MPI --map-by socket
2	1,407461966	1,351407942	0,826019986	1,350711325
4	2,588420344	2,525103441	0,790827699	2,145472624
8	4,352400318	2,627263934	0,51930996	2,357546916
16	7,145666147	1,891711472	0,280478638	1,909437376
32	5,789021101	0,141216948	0,143032085	0,141276546

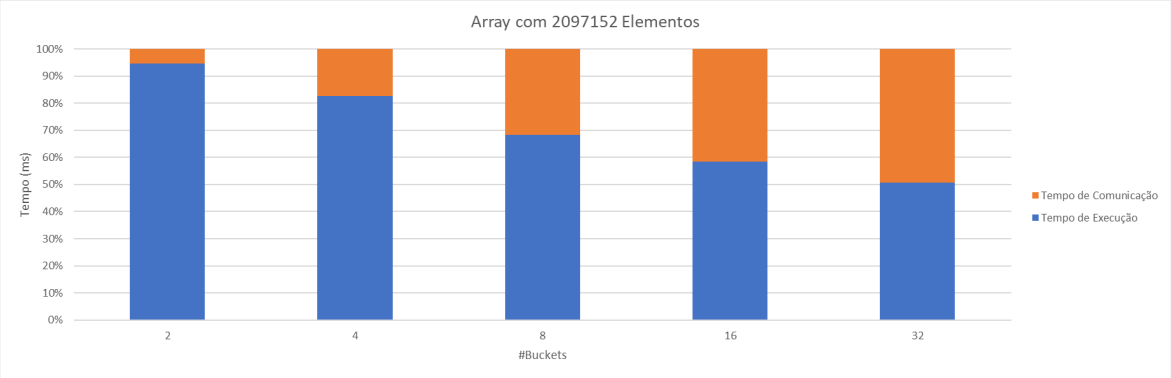
F Custos de Comunicação

F.1 Por Core

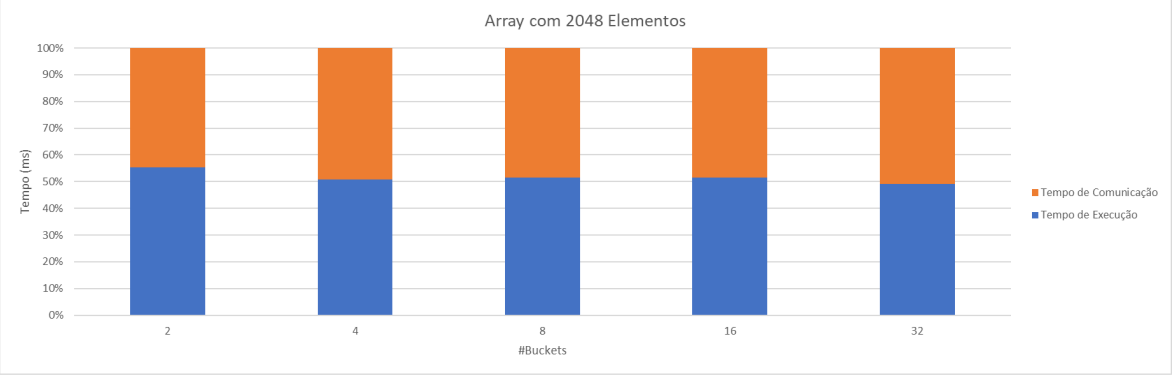


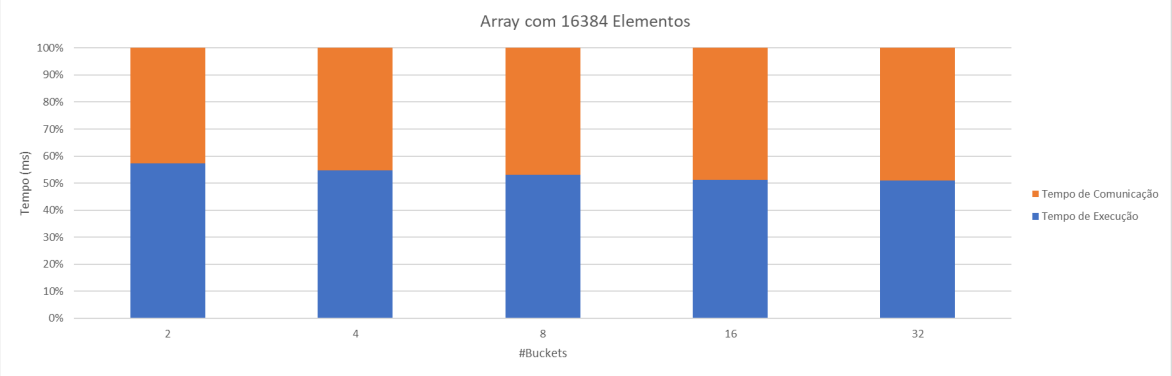


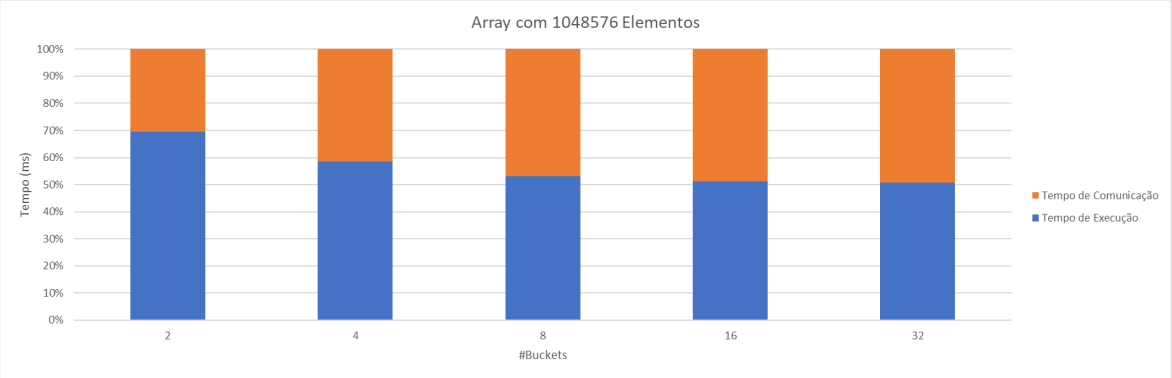


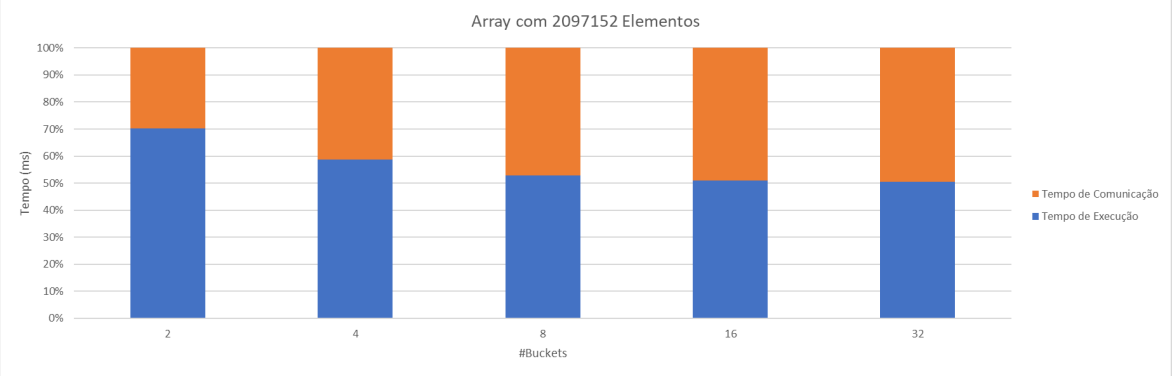

Array com 2097152 Elementos
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
Tempo (ms)
2
4
8
16
32
#Buckets
Tempo de Comunicação
Tempo de Execução

F.2 Por Nodo

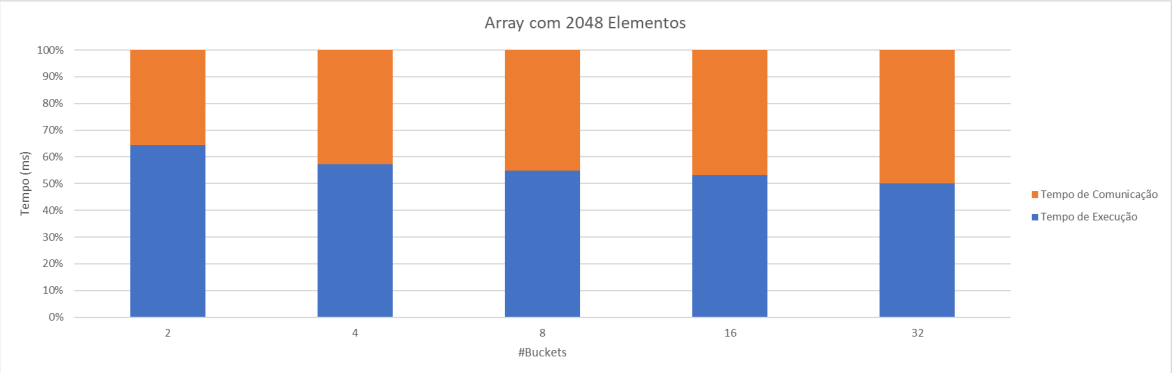

Array com 2048 Elementos
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
Tempo (ms)
2
4
8
16
32
#Buckets
Tempo de Comunicação
Tempo de Execução

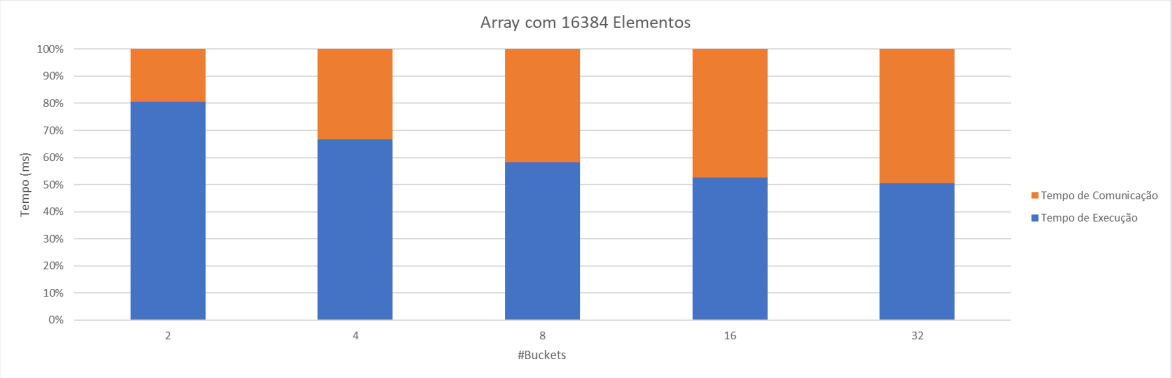

Array com 16384 Elementos
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
Tempo (ms)
2
4
8
16
32
#Buckets
Tempo de Comunicação
Tempo de Execução

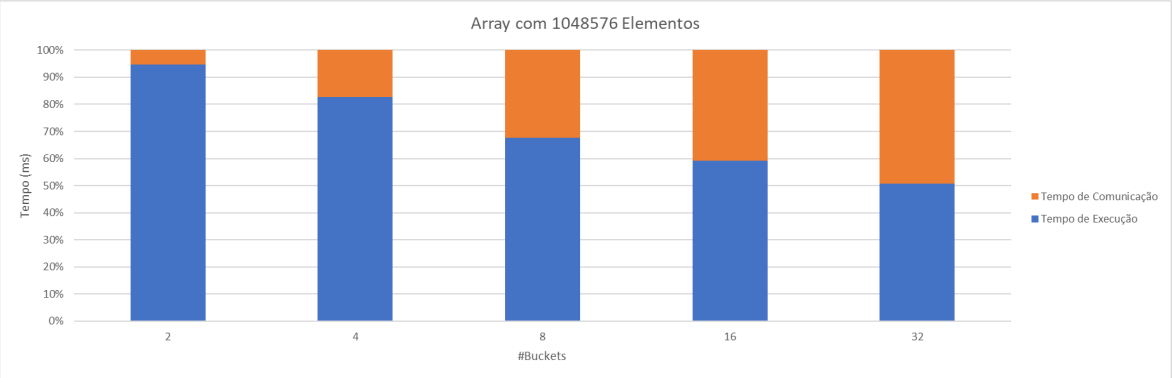


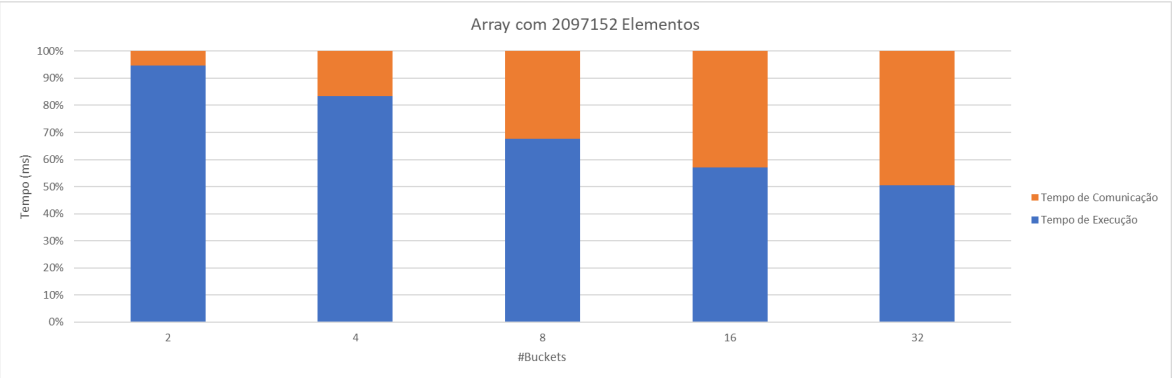


F.3 Por Socket

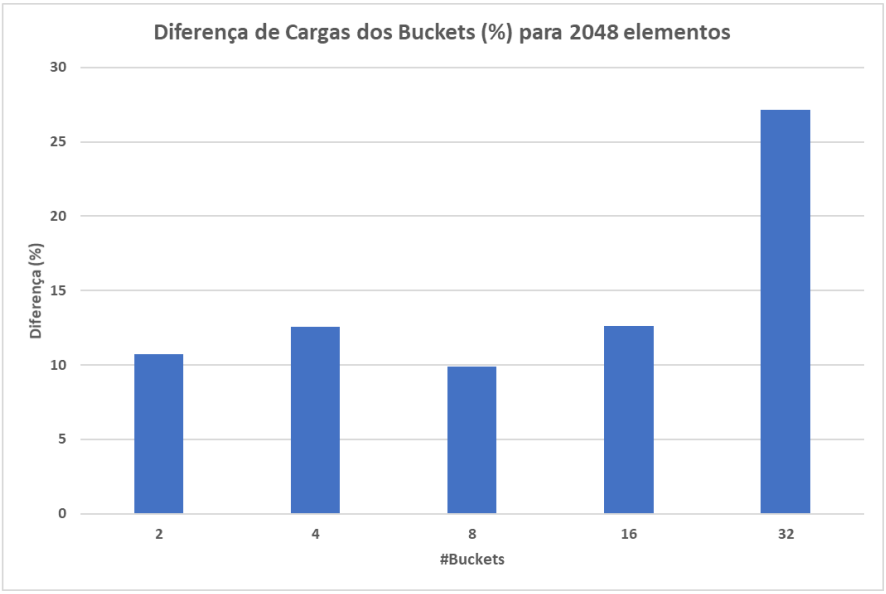


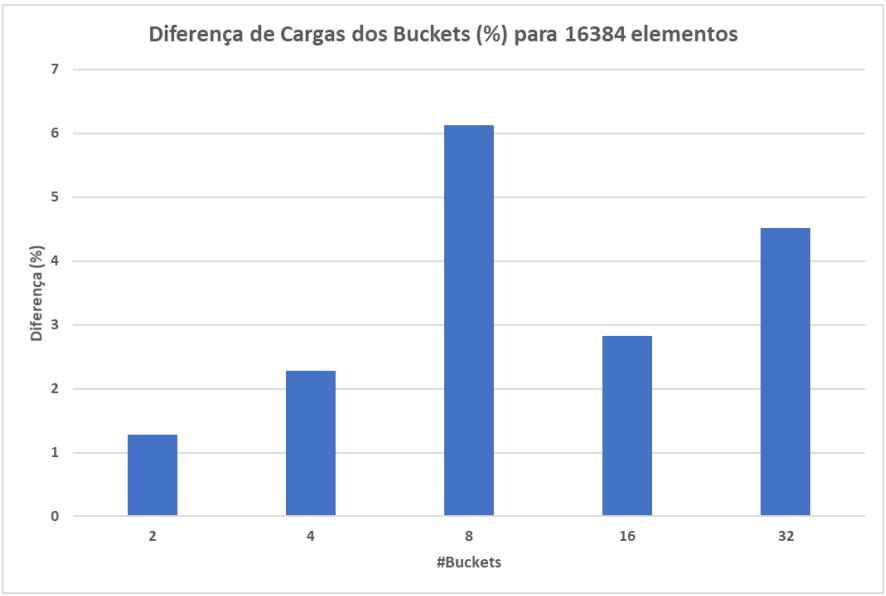

Array com 16384 Elementos
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
Tempo (ms)
2
4
8
16
32
#Buckets
Tempo de Comunicação
Tempo de Execução

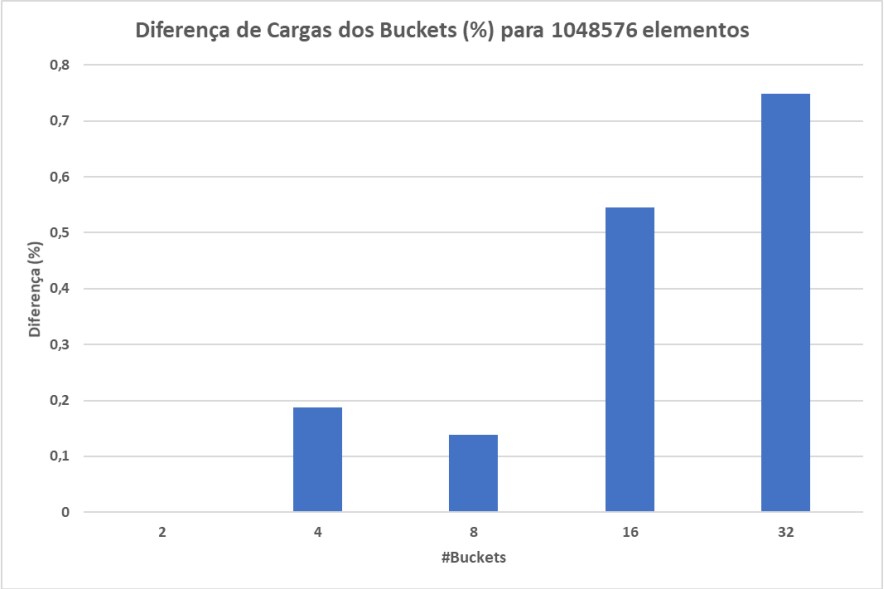

Array com 1048576 Elementos
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
Tempo (ms)
2
4
8
16
32
#Buckets
Tempo de Comunicação
Tempo de Execução

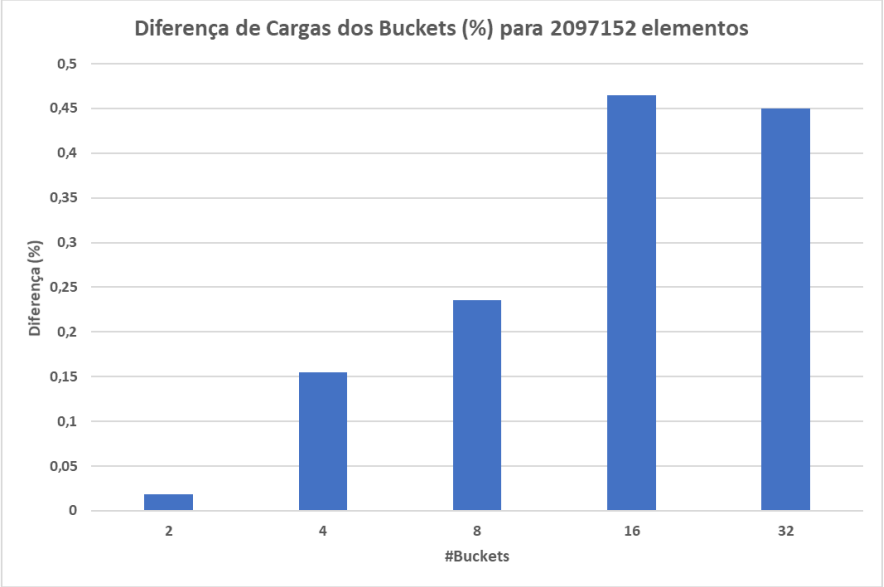

Array com 2097152 Elementos
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
Tempo (ms)
2
4
8
16
32
#Buckets
Tempo de Comunicação
Tempo de Execução

G Grau de Distribuição da Carga entre Processos






Diferença de Cargas dos Buckets (%) para 1048576 elementos
Diferença (%)
0,8
0,7
0,6
0,5
0,4
0,3
0,2
0,1
0
2
4
8
16
32
#Buckets


Diferença de Cargas dos Buckets (%) para 2097152 elementos
Diferença (%)
0,5
0,45
0,4
0,35
0,3
0,25
0,2
0,15
0,1
0,05
0
2
4
8
16
32
#Buckets