

Comparação de Tecnologias de Comunicação entre Clusters no Processador MPPA-256: Um Estudo com Aplicações do CAP Benchmarks

David Grunheidt Vilela Ordine, Márcio Castro

Departamento de Informática e Estatística (INE)
Universidade Federal de Santa Catarina (UFSC)
`david.ordine@grad.ufsc.br`
`marcio.castro@ufsc.br`

27 de dezembro de 2020

Sumário

Sumário

Introdução

Fundamentação Teórica

MPPA-256
OpenMP API
Modelo mestre-escravo
MPPA IPC API
MPPA ASYNC API
CAP-Benchmarks

Desenvolvimento

Friendly Numbers
Fast
Gaussian Filter
LU Factorization
K-Means
Integer-Sort

Resultados

Desempenho
Tráfego de dados
Consumo energético

Conclusão

- Introdução
- Fundamentação Teórica
- Desenvolvimento
- Resultados
- Conclusão

Contexto

Sumário

Introdução

Fundamentação Teórica

MPPA-256

OpenMP API

Modelo mestre-escravo

MPPA IPC API

MPPA ASYNC API

CAP-Benchmarks

Desenvolvimento

Friendly Numbers

Fast

Gaussian Filter

LU Factorization

K-Means

Integer-Sort

Resultados

Desempenho

Tráfego de dados

Consumo energético

Conclusão



- Objetivo de alcançar a computação em *Exascale*
- Viável - 50 GFlops/W X Máximo - 14.719 GFlops/W
- *Trade-off* desproporcional entre desempenho e consumo energético (barreira de potência)

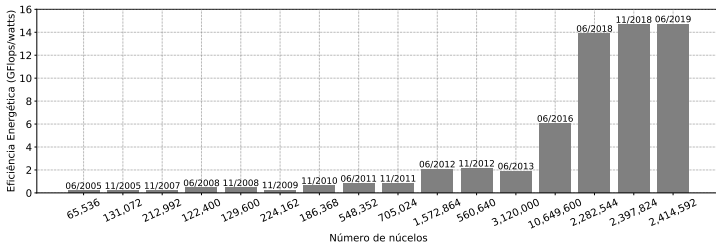


Figura 1: Eficiência energética X número de núcleos do TOP1 supercomputador do ranking TOP500 ao passar dos anos.

Sumário

Introdução

Fundamentação Teórica

MPPA-256
OpenMP API
Modelo mestre-escravo
MPPA IPC API
MPPA ASYNC API
CAP-Benchmarks

Desenvolvimento

Friendly Numbers
Fast
Gaussian Filter
LU Factorization
K-Means
Integer-Sort

Resultados

Desempenho
Tráfego de dados
Consumo energético

Conclusão

- Processadores *manycore* de baixa potência como solução do problema
 - MPPA-256 [1]
 - Adapteva Epiphany [4]
 - SW26010 (*Sunway TaihuLight*) [2]
- MPPA-256 como objeto de estudo deste trabalho
- Necessário avaliar seu desempenho:
 - CAP-Benchmarks

Sumário

Introdução

Fundamentação Teórica

MPPA-256
OpenMP API
Modelo mestre-escravo
MPPA IPC API
MPPA ASYNC API
CAP-Benchmarks

Desenvolvimento

Friendly Numbers
Fast
Gaussian Filter
LU Factorization
K-Means
Integer-Sort

Resultados

Desempenho
Tráfego de dados
Consumo energético

Conclusão

- Duas versões do *benchmark*:
 - *MPPA Interprocess Communication API* (IPC)
 - *MPPA Asynchronous Communication API* (ASYNC)
- **Objetivos:**
 - Portar aplicações do CAP-Bench para a API ASYNC
 - Avaliar os custos e benefícios do MPPA-256 (tempos de execução, gasto energético e tráfego de dados)
 - Comparar as tecnologias IPC e ASYNC
 - Estudar as peculiaridades de cada tecnologia

Sumário

Introdução

Fundamentação Teórica

MPPA-256

OpenMP API

Modelo mestre-escravo

MPPA IPC API

MPPA ASYNC API

CAP-Benchmarks

Desenvolvimento

Friendly Numbers

Fast

Gaussian Filter

LU Factorization

K-Means

Integer-Sort

Resultados

Desempenho

Tráfego de dados

Consumo energético

Conclusão

- 16 *clusters* de computação (CCs)
 - 16 núcleos em cada CC, atuando a 400 Mhz cada
 - Memória compartilhada de 2MB
 - Memória cache associativa 2-way, privadas e de 32 kB
- 4 *clusters* de E/S
 - 4 núcleos gerenciadores de recursos em cada

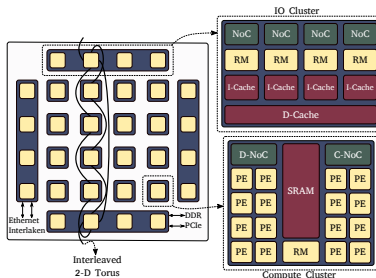


Figura 2: Visão arquitetural do MPPA-256 [5]

Open Multi-Processing

Sumário

Introdução

Fundamentação Teórica

MPPA-256

OpenMP API

Modelo mestre-escravo

MPPA IPC API

MPPA ASYNC API

CAP-Benchmarks

Desenvolvimento

Friendly Numbers

Fast

Gaussian Filter

LU Factorization

K-Means

Integer-Sort

Resultados

Desempenho

Tráfego de dados

Consumo energético

Conclusão

- Implementação de aplicações paralelas
 - C, C++ e Fortran
- Voltada ao *multithreading*
- Centrada no modelo Fork-Join

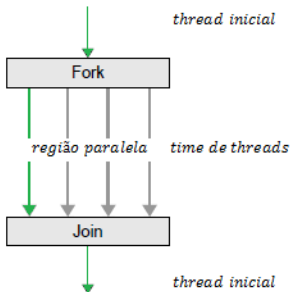


Figura 3: Esquema do modelo fork-join.

Open Multi-Processing

Sumário

Introdução

Fundamentação Teórica

MPPA-256

OpenMP API

Modelo mestre-escravo

MPPA IPC API

MPPA ASYNC API

CAP-Benchmarks

Desenvolvimento

Friendly Numbers

Fast

Gaussian Filter

LU Factorization

K-Means

Integer-Sort

Resultados

Desempenho

Tráfego de dados

Consumo energético

Conclusão



■ Diretivas de compilação e regiões paralelas

■ #pragma omp parallel for

```
1  static void createArray() {  
2      int array[max], index;  
3      int index_max = 10;  
4      #pragma omp parallel for  
5      for (index = 0; i < index_max; index++)  
6          array[index] = index;  
7  }
```

Figura 4: Execução de um loop de forma paralela.

Open Multi-Processing

Sumário

Introdução

Fundamentação Teórica

MPPA-256

OpenMP API

Modelo mestre-escravo

MPPA IPC API

MPPA ASYNC API

CAP-Benchmarks

Desenvolvimento

Friendly Numbers

Fast

Gaussian Filter

LU Factorization

K-Means

Integer-Sort

Resultados

Desempenho

Tráfego de dados

Consumo energético

Conclusão



■ Cláusulas OpenMP

■ private, default e reduction

```
1 static int partial_friendly_sum = 0;
2 ...
3 static void countFriends() {
4     int i; /* Loop indexes. */
5
6     #pragma omp parallel for private(i) default(shared) reduction(+: partial_friendly_sum)
7     for (i = offset; i < offset + tasksize; i++) {
8         for (int j = 0; j < i; j++) {
9             if ((allTasks[i].num == allTasks[j].num) && (allTasks[i].den == allTasks[j].den))
10                 partial_friendly_sum++;
11         }
12     }
13 }
```

Figura 5: Exemplo de leitura e armazenamento seguro em variável compartilhada entre threads em uma das aplicações do CAP-Bench.

Modelo mestre-escravo no MPPA-256

Sumário

Introdução

Fundamentação Teórica

MPPA-256

OpenMP API

Modelo mestre-escravo

MPPA IPC API

MPPA ASYNC API

CAP-Benchmarks

Desenvolvimento

Friendly Numbers

Fast

Gaussian Filter

LU Factorization

K-Means

Integer-Sort

Resultados

Desempenho

Tráfego de dados

Consumo energético

Conclusão



- *Clusters* de E/S gerenciam os *clusters* de computação
- Execução de códigos binários diferentes
- Cada cluster executa um código paralelizado com OpenMP

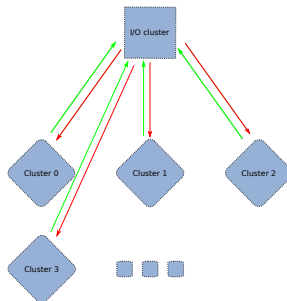


Figura 6: Fluxo de uma aplicação seguindo o modelo mestre-escravo no MPPA-256.

MPPA IPC API

Sumário

Introdução

Fundamentação Teórica

MPPA-256

OpenMP API

Modelo mestre-escravo

MPPA IPC API

MPPA ASYNC API

CAP-Benchmarks

Desenvolvimento

Friendly Numbers

Fast

Gaussian Filter

LU Factorization

K-Means

Integer-Sort

Resultados

Desempenho

Tráfego de dados

Consumo energético

Conclusão



- Definir caminhos de comunicação explicitamente
- Controle dos caminhos é feito pelo programador
- Conhecimento profundo da arquitetura

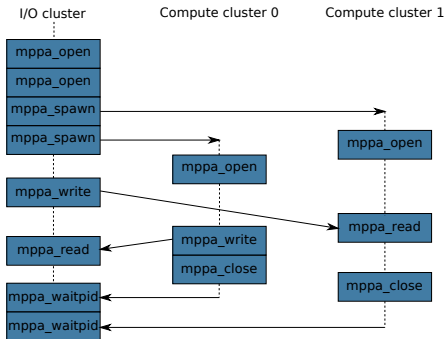


Figura 7: Fluxo de uma aplicação usando funções do tipo POSIX da IPC no MPPA-256.

MPPA ASYNC API [3]

Sumário

Introdução

Fundamentação Teórica

MPPA-256

OpenMP API

Modelo mestre-escravo

MPPA IPC API

MPPA ASYNC API

CAP-Benchmarks

Desenvolvimento

Friendly Numbers

Fast

Gaussian Filter

LU Factorization

K-Means

Integer-Sort

Resultados

Desempenho

Tráfego de dados

Consumo energético

Conclusão

- Comunicação assíncrona e unilateral entre *clusters*
- Segmentos sobre espaços locais de memória
 - Abstração do modelo de memória distribuída
 - Clonagem de segmentos
 - Operações *put/get* sobre segmentos

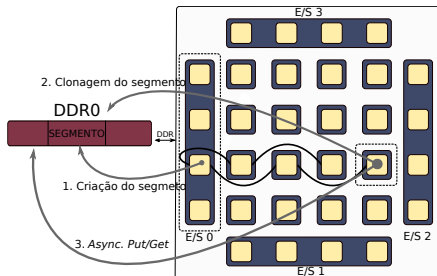


Figura 8: Visão esquemática do funcionamento da biblioteca async

CAP-Benchmarks [6]

Sumário

Introdução

Fundamentação Teórica

MPPA-256

OpenMP API

Modelo mestre-escravo

MPPA IPC API

MPPA ASYNC API

CAP-Benchmarks

Desenvolvimento

Friendly Numbers

Fast

Gaussian Filter

LU Factorization

K-Means

Integer-Sort

Resultados

Desempenho

Tráfego de dados

Consumo energético

Conclusão

- Conjunto de 7 aplicações desenvolvidas em C
 - Uma das aplicações não foi portada neste trabalho
- Domínios de problemas variados
- Diferentes padrões paralelos
- Testar o processador em diversas condições
- Criado para avaliar diferentes processadores de arquitetura *Manycore*

Sumário

Introdução

Fundamentação Teórica

MPPA-256
OpenMP API
Modelo mestre-escravo
MPPA IPC API
MPPA ASYNC API
CAP-Benchmarks

Desenvolvimento

Friendly Numbers
Fast
Gaussian Filter
LU Factorization
K-Means
Integer-Sort

Resultados

Desempenho
Tráfego de dados
Consumo energético

Conclusão

- Fluxo de desenvolvimento de cada aplicação
 - Alterações de otimização
 - Alterações de tecnologia de comunicação
- Otimizações valem para ambas as versões
- Comparação dos resultados leva em conta somente a tecnologia

Friendly Numbers

Sumário

Introdução

Fundamentação Teórica

MPPA-256

OpenMP API

Modelo mestre-escravo

MPPA IPC API

MPPA ASYNC API

CAP-Benchmarks

Desenvolvimento

Friendly Numbers

Fast

Gaussian Filter

LU Factorization

K-Means

Integer-Sort

Resultados

Desempenho

Tráfego de dados

Consumo energético

Conclusão

- Dois números naturais são amigáveis se compartilham a mesma abundância
- Abundância A de um número $n \rightarrow A(n) = \frac{\sigma(n)}{n}$
 - $\sigma(n)$ representa a soma de todos os divisores de n
- Calcula e compara as abundâncias de números em um intervalo
- Quais pares de números são amigáveis neste intervalo?

Alterações no FN

Sumário

Introdução

Fundamentação Teórica

MPPA-256

OpenMP API

Modelo mestre-escravo

MPPA IPC API

MPPA ASYNC API

CAP-Benchmarks

Desenvolvimento

Friendly Numbers

Fast

Gaussian Filter

LU Factorization

K-Means

Integer-Sort

Resultados

Desempenho

Tráfego de dados

Consumo energético

Conclusão



- Segmento sobre o *array* de tarefas
- Operações de PUT/GET somente nos *slaves*
- Otimização na função de soma dos divisores
 - Antes: loop entre 2 e o número
 - Agora: loop entre 2 e metade do número
- POSIX *Threads* -> OpenMP
 - Implementação do paralelismo reduzido: 50 linhas -> 5.

```
1  /* Items to be sent to slaves */
2  typedef struct {
3      int number; /* Number */
4      int num; /* Numerator */
5      int den; /* Denominator */
6  } Item;
7
8  #define MAX_TASK_SIZE 65536
9
10 static Item tasks[MAX_TASK_SIZE];
```

Figura 9: Definição das tarefas por parte do cluster de IO.

Features from Accelerated Segment Test

Sumário

Introdução

Fundamentação Teórica

MPPA-256

OpenMP API

Modelo mestre-escravo

MPPA IPC API

MPPA ASYNC API

CAP-Benchmarks

Desenvolvimento

Friendly Numbers

Fast

Gaussian Filter

LU Factorization

K-Means

Integer-Sort

Resultados

Desempenho

Tráfego de dados

Consumo energético

Conclusão

- Algoritmo de detecção de cantos
- Extrair pontos importantes em uma imagem
- Círculo de 16 *pixels* testa se um ponto p é um canto
- Brilho de um *pixel* é um inteiro entre 1 e 16
- É canto se:
 - Brilho dos N *pixels* do círculo $>$ brilho de p + limiar t
 - Brilho dos N *pixels* do círculo $<$ brilho de p - limiar t

Alterações no FAST

Sumário

Introdução

Fundamentação Teórica

MPPA-256

OpenMP API

Modelo mestre-escravo

MPPA IPC API

MPPA ASYNC API

CAP-Benchmarks

Desenvolvimento

Friendly Numbers

Fast

Gaussian Filter

LU Factorization

K-Means

Integer-Sort

Resultados

Desempenho

Tráfego de dados

Consumo energético

Conclusão



- Versão antiga
 - Tarefas enviadas explicitamente aos CCs
 - CCs aguardavam em um `while(true)`
 - Mensagens sinalizando continuação ou finalização
- Nova versão
 - CCs sabem exatamente a quantidade de tarefas a serem realizadas
 - Solicitam tarefas ao *cluster* de I/O
 - Mensagens do *master* para um *slave*: Redução 4 -> 1
 - Mensagens de um *slave* para o *master*: Redução 2 -> 1
- Inversão da lógica de tarefas garante grande paralelismo

Gaussian Filter

Sumário

Introdução

Fundamentação Teórica

MPPA-256

OpenMP API

Modelo mestre-escravo

MPPA IPC API

MPPA ASYNC API

CAP-Benchmarks

Desenvolvimento

Friendly Numbers

Fast

Gaussian Filter

LU Factorization

K-Means

Integer-Sort

Resultados

Desempenho

Tráfego de dados

Consumo energético

Conclusão

- Algoritmo que aplica um filtro de suavização em uma imagem
- Poucas alterações entre versões do CAP-Bench
- Dois segmentos criados
 - No *array* que guarda a máscara a ser aplicada
 - No *array* que guarda um pedaço da imagem a ser processado (*chunk*)
- Um acesso por vez ao segundo segmento
 - Memória insuficiente para criar um segmento sobre toda a imagem
- Operações de PUT/GET somente nos *slaves*

LU Factorization

Sumário

Introdução

Fundamentação Teórica

MPPA-256

OpenMP API

Modelo mestre-escravo

MPPA IPC API

MPPA ASYNC API

CAP-Benchmarks

Desenvolvimento

Friendly Numbers

Fast

Gaussian Filter

LU Factorization

K-Means

Integer-Sort

Resultados

Desempenho

Tráfego de dados

Consumo energético

Conclusão



- Aplicação com maior número de alterações
- Transforma uma matriz A em duas matrizes
 - Uma triangular inferior L
 - Uma triangular superior U
 - $A = L * U$

$$\begin{bmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ L_{10} & 1 & 0 \\ L_{20} & L_{21} & 1 \end{bmatrix} \begin{bmatrix} U_{00} & U_{01} & U_{02} \\ 0 & U_{11} & U_{12} \\ 0 & 0 & U_{22} \end{bmatrix}$$

Lower Triangular

Upper Triangular

Figura 10: Resultado esperado pelo algoritmo LU.

LU - Versão Antiga

Sumário

Introdução

Fundamentação Teórica

MPPA-256

OpenMP API

Modelo mestre-escravo

MPPA IPC API

MPPA ASYNC API

CAP-Benchmarks

Desenvolvimento

Friendly Numbers

Fast

Gaussian Filter

LU Factorization

K-Means

Integer-Sort

Resultados

Desempenho

Tráfego de dados

Consumo energético

Conclusão



- Permutações de linhas e colunas
- $PLU \rightarrow P * A = L * U$
- P é o agregado de todas as permutações
- Envio de blocos da matriz A de forma errada

$$A = \begin{bmatrix} 1 & 5 & 3 & 8 \\ 12 & 7 & 15 & 12 \\ 3 & 12 & 92 & 8 \\ 6 & 9 & 3 & 1 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 5 & 3 & 8 \\ 12 & 7 & 15 & 12 \\ 3 & 12 & 92 & 8 \\ 6 & 9 & 3 & 1 \end{bmatrix}$$

Figura 11: Exemplo de um bloco passado a um slave em cada versão da LU. (A = Versão antiga, B = Versão nova)

Sumário

Introdução

Fundamentação Teórica

MPPA-256
OpenMP API
Modelo mestre-escravo
MPPA IPC API
MPPA ASYNC API
CAP-Benchmarks

Desenvolvimento

Friendly Numbers
Fast
Gaussian Filter
LU Factorization
K-Means
Integer-Sort

Resultados

Desempenho
Tráfego de dados
Consumo energético

Conclusão

- Remoção das permutações de linhas e colunas
- Correção no envio de blocos aos *slaves*
- Segmentos sobre o *array* de tarefas e sobre a matriz
- CCs consultam segmento de tarefas
 - Verificam se continuam ou finalizam a execução
 - Informação anterior possibilita acessar blocos na matriz

K-Means

Sumário

Introdução

Fundamentação Teórica

MPPA-256

OpenMP API

Modelo mestre-escravo

MPPA IPC API

MPPA ASYNC API

CAP-Benchmarks

Desenvolvimento

Friendly Numbers

Fast

Gaussian Filter

LU Factorization

K-Means

Integer-Sort

Resultados

Desempenho

Tráfego de dados

Consumo energético

Conclusão



- Algoritmo de clusterização de dados
- Dados n pontos de dimensão d
 - Particionar estes pontos em k conjuntos

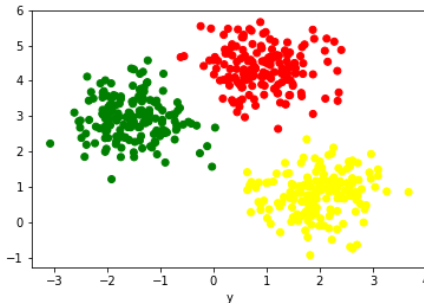


Figura 12: Exemplo genérico do resultado final da execução do K-means plotado em um gráfico.

Alterações no K-Means

Sumário

Introdução

Fundamentação Teórica

MPPA-256
OpenMP API
Modelo mestre-escravo
MPPA IPC API
MPPA ASYNC API
CAP-Benchmarks

Desenvolvimento

Friendly Numbers
Fast
Gaussian Filter
LU Factorization
K-Means
Integer-Sort

Resultados

Desempenho
Tráfego de dados
Consumo energético

Conclusão



- Simplificação no modelo de dados
- Versão antiga
 - Pontos eram *structs* -> {dimensão, elementos}
- Versão atual
 - Pontos salvos em um *array* unidimensional
 - Tamanho do array = nPontos * dimensão
 - Facilidade para criar o segmento sobre os pontos

Alterações no K-Means

Sumário

Introdução

Fundamentação Teórica

MPPA-256

OpenMP API

Modelo mestre-escravo

MPPA IPC API

MPPA ASYNC API

CAP-Benchmarks

Desenvolvimento

Friendly Numbers

Fast

Gaussian Filter

LU Factorization

K-Means

Integer-Sort

Resultados

Desempenho

Tráfego de dados

Consumo energético

Conclusão

- Alteração no cálculo dos *centroids*
 - Média dos elementos de um certo grupo k
- Versão antiga
 - Cálculo todo feito nos CCs
 - CCs enviavam as populações parciais ao *master*
 - *Master* respondia a soma desses dados
- Versão atual
 - Somente a soma dos vetores parciais é feita nos CCs
 - População total facilmente acessível no *master*
 - Cálculo da média é feita no *cluster* de E/S
 - Redução de 2 operações de escrita e 2 operações de leitura

Integer Sort

Sumário

Introdução

Fundamentação Teórica

MPPA-256
OpenMP API
Modelo mestre-escravo
MPPA IPC API
MPPA ASYNC API
CAP-Benchmarks

Desenvolvimento

Friendly Numbers
Fast
Gaussian Filter
LU Factorization
K-Means
Integer-Sort

Resultados

Desempenho
Tráfego de dados
Consumo energético

Conclusão

- Algoritmo de ordenação de um grande número de inteiros
- CAP-Bench -> *bucket-sort*
- Elementos a serem ordenados são divididos em baldes
 - Armazena certa quantidade de inteiros

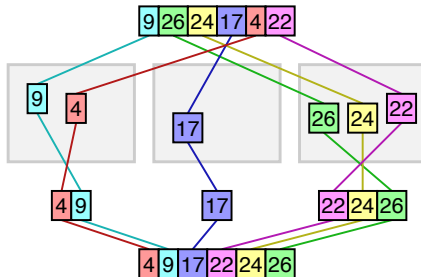


Figura 13: Exemplo genérico do resultado final da execução do IS na variação bucket-sort.

Alterações no IS

Sumário

Introdução

Fundamentação Teórica

MPPA-256

OpenMP API

Modelo mestre-escravo

MPPA IPC API

MPPA ASYNC API

CAP-Benchmarks

Desenvolvimento

Friendly Numbers

Fast

Gaussian Filter

LU Factorization

K-Means

Integer-Sort

Resultados

Desempenho

Tráfego de dados

Consumo energético

Conclusão



■ Versão antiga

- Ordenação parcial dos *buckets* não otimizada
- Número de elementos ordenados é sempre o número máximo de elementos em um *bucket* na maior classe de problema (*huge*)
- Valor escolhido pois é divisível por 16 e abrange todas as classes
- Grande quantidade de elementos *dummy*

■ Versão nova

- Ordenação parcial dos *buckets* otimizada
- Número de elementos ordenados = número de elementos de um *bucket* + no máximo 16 elementos
- Continua sendo divisível por 16 e abrangendo todas as classes
- Quantidade insignificante de elementos *dummy*

Métricas definidas

Sumário

Introdução

Fundamentação Teórica

MPPA-256
OpenMP API
Modelo mestre-escravo
MPPA IPC API
MPPA ASYNC API
CAP-Benchmarks

Desenvolvimento

Friendly Numbers
Fast
Gaussian Filter
LU Factorization
K-Means
Integer-Sort

Resultados

Desempenho
Tráfego de dados
Consumo energético

Conclusão

Métrica	Unidade
Média do tempo de execução dos <i>slaves</i>	Segundos
Tempo de execução do processo <i>master</i>	Segundos
Tempo total de comunicação entre <i>master</i> e <i>slaves</i>	Segundos
Quantidade de dados que o <i>master</i> envia aos <i>slaves</i>	Megabytes
Quantidade de dados que o <i>master</i> recebe dos <i>slaves</i>	Megabytes
Potência média durante a execução da aplicação	Watts
Consumo de energia durante a execução da aplicação	joules

Tabela 1: Métricas definidas para comparação das tecnologias.

Variações de execução

Sumário

Introdução

Fundamentação Teórica

MPPA-256
OpenMP API
Modelo mestre-escravo
MPPA IPC API
MPPA ASYNC API
CAP-Benchmarks

Desenvolvimento

Friendly Numbers
Fast
Gaussian Filter
LU Factorization
K-Means
Integer-Sort

Resultados

Desempenho
Tráfego de dados
Consumo energético

Conclusão

- Classes de tamanho *tiny*, *small*, *standard*, *large* e *huge*
- Para cada classe, variou-se o número de *clusters* de computação entre 1, 2, 4, 8 e 16
- 5 repetições para cada uma dessas variações
- 16 núcleos de processamento ativos em todas as variações

Significado das tabelas

Sumário

Introdução

Fundamentação Teórica

MPPA-256

OpenMP API

Modelo mestre-escravo

MPPA IPC API

MPPA ASYNC API

CAP-Benchmarks

Desenvolvimento

Friendly Numbers

Fast

Gaussian Filter

LU Factorization

K-Means

Integer-Sort

Resultados

Desempenho

Tráfego de dados

Consumo energético

Conclusão

- Exibem redução ou aumento de determinada métrica
 - $((ResultadoASYNC/ResultadoIPC) - 1) * 100$
 - Porcentagem de redução/aumento na versão ASYNC
- Focam no melhor e pior resultado de redução

Tempos de execução de cada CC

Sumário

Introdução

Fundamentação Teórica

MPPA-256

OpenMP API

Modelo mestre-escravo

MPPA IPC API

MPPA ASYNC API

CAP-Benchmarks

Desenvolvimento

Friendly Numbers

Fast

Gaussian Filter

LU Factorization

K-Means

Integer-Sort

Resultados

Desempenho

Tráfego de dados

Consumo energético

Conclusão



■ Melhoria em todos os *kernels*

■ Redução variou entre 57.14% e 94.44%

App	Redução Mínima					Redução Máxima				
	Classe	NClusters	IPC(s)	ASYNC(s)	Redução(%)	Classe	NClusters	IPC(s)	ASYNC(s)	Redução(%)
FAST	Tiny	8	0.72	0.05	93.06	Tiny	16	0.36	0.02	94.44
FN	Huge	1	34412.07	1952.39	94.33	Small	16	267.9	15.14	94.35
GF	Tiny	4	1.38	0.09	93.48	Tiny	8	0.69	0.04	94.2
IS	Tiny	8	1.82	0.78	57.14	Huge	1	282.06	115.03	59.22
KM	Tiny	16	2.86	0.19	93.36	Standard	1	1427.49	91.55	93.59
LU	Tiny	2	0.28	0.09	67.86	Huge	2	40.15	9.02	77.53

Tabela 2: Reduções ao comparar-se os tempos dos processos slaves.

Tempos de execução do processo *master*

Sumário

Introdução

Fundamentação Teórica

MPPA-256

OpenMP API

Modelo mestre-escravo

MPPA IPC API

MPPA ASYNC API

CAP-Benchmarks

Desenvolvimento

Friendly Numbers

Fast

Gaussian Filter

LU Factorization

K-Means

Integer-Sort

Resultados

Desempenho

Tráfego de dados

Consumo energético

Conclusão



- Melhoria em todos os *kernels*
 - Exceção: LU
- Redução variou entre -28.79% e 70.2%
- FAST e GF com tempo 0 em ambas versões

App	Redução Mínima					Redução Máxima				
	Classe	NClusters	IPC(s)	ASYNC(s)	Redução(%)	Classe	NClusters	IPC(s)	ASYNC(s)	Redução(%)
FN	Tiny	1	0.14	0.08	42.86	Huge	1	118.54	35.32	70.2
IS	Standard	1	12.27	10.96	10.68	Huge	8	56.72	46.5	18.02
KM	Tiny	1	0.01	0.01	0.0	Tiny	4	0.02	0.01	50.0
LU	Huge	2	0.66	0.85	-28.79	Tiny	2	0.02	0.02	0.0

Tabela 3: Reduções ao comparar-se os tempos do processo master.

Tempos de comunicação

Sumário

Introdução

Fundamentação Teórica

MPPA-256

OpenMP API

Modelo mestre-escravo

MPPA IPC API

MPPA ASYNC API

CAP-Benchmarks

Desenvolvimento

Friendly Numbers

Fast

Gaussian Filter

LU Factorization

K-Means

Integer-Sort

Resultados

Desempenho

Tráfego de dados

Consumo energético

Conclusão



■ Melhoria em todos os *kernels*

■ Exceção: IS

■ Redução variou entre -213.16% até 99.66%

App	Redução Mínima					Redução Máxima				
	Classe	NClusters	IPC(s)	ASYNC(s)	Redução(%)	Classe	NClusters	IPC(s)	ASYNC(s)	Redução(%)
FAST	Tiny	16	1.44	1.16	19.44	Huge	8	109.55	1.25	98.86
FN	Huge	1	34412.4	976.35	97.16	Huge	16	2159.8	7.34	99.66
GF	Tiny	16	1.31	1.1	16.03	Huge	4	1162.81	20.57	98.23
IS	Tiny	16	1.14	3.57	-213.16	Huge	1	286.46	80.34	71.95
KM	Tiny	16	3.82	1.11	70.94	Huge	16	670.86	4.28	99.36
LU	Tiny	16	7.22	1.36	81.16	Large	8	193.78	6.2	96.8

Tabela 4: Reduções ao comparar-se os tempos de comunicação.

Tráfego de dados

Sumário

Introdução

Fundamentação Teórica

MPPA-256

OpenMP API

Modelo mestre-escravo

MPPA IPC API

MPPA ASYNC API

CAP-Benchmarks

Desenvolvimento

Friendly Numbers

Fast

Gaussian Filter

LU Factorization

K-Means

Integer-Sort

Resultados

Desempenho

Tráfego de dados

Consumo energético

Conclusão



- Maioria dos resultados foram iguais em ambas versões
 - Para ambas as métricas de tráfego de dados
- KM:
 - Redução entre 6.67% e 48.51% para os dados enviados
 - Redução entre -1.56% e 19.83% para os dados recebidos
- LU:
 - Resultados iguais para todas as classes exceto a *Large*
 - Redução de -100% nos dados enviados e -171.88% nos dados recebidos
- Possível indicação de *bugs*

Potência média

Sumário

Introdução

Fundamentação Teórica

MPPA-256

OpenMP API

Modelo mestre-escravo

MPPA IPC API

MPPA ASYNC API

CAP-Benchmarks

Desenvolvimento

Friendly Numbers

Fast

Gaussian Filter

LU Factorization

K-Means

Integer-Sort

Resultados

Desempenho

Tráfego de dados

Consumo energético

Conclusão



- Redução variou entre -132.57% até 3.75%
- ASYNC necessita de mais recursos ativos

App	Redução Mínima					Redução Máxima				
	Classe	NClusters	IPC(W)	ASYNC(W)	Redução(%)	Classe	NClusters	IPC(W)	ASYNC(W)	Redução(%)
FAST	Small	16	4.52	5.14	-13.72	Small	1	4.27	4.11	3.75
FN	Large	16	4.79	10.25	-113.99	Huge	1	4.28	4.76	-11.21
GF	Huge	16	4.84	5.91	-22.11	Small	1	4.27	4.17	2.34
IS	Large	16	4.48	5.99	-33.71	Large	1	4.28	4.41	-3.04
KM	Huge	16	4.82	11.21	-132.57	Small	1	4.25	4.71	-10.82
LU	Huge	16	4.42	6.05	-36.88	Small	1	4.23	4.09	3.31

Tabela 5: Reduções ao comparar-se a potência média durante execução.

Consumo energético

Sumário

Introdução

Fundamentação Teórica

MPPA-256

OpenMP API

Modelo mestre-escravo

MPPA IPC API

MPPA ASYNC API

CAP-Benchmarks

Desenvolvimento

Friendly Numbers

Fast

Gaussian Filter

LU Factorization

K-Means

Integer-Sort

Resultados

Desempenho

Tráfego de dados

Consumo energético

Conclusão

- Redução variou entre -30.31% e 93.64%
- Menor tempo de execução justifica maior potência média

App	Redução Mínima					Redução Máxima				
	Classe	NClusters	IPC(J)	ASYNC(J)	Redução(%)	Classe	NClusters	IPC(J)	ASYNC(J)	Redução(%)
FAST	Tiny	16	38.79	43.92	-13.23	Huge	1	3759.55	456.35	87.86
FN	Tiny	16	666.97	129.65	80.56	Standard	1	36549.92	2325.1	93.64
GF	Tiny	16	38.4	44.64	-16.25	Huge	1	20504.65	1783.83	91.3
IS	Tiny	16	52.2	68.02	-30.31	Large	1	751.9	395.02	47.46
KM	Tiny	16	47.55	46.21	2.82	Standard	1	6131.17	460.56	92.49
LU	Tiny	16	64.68	44.14	31.76	Huge	8	1535.77	103.8	93.24

Tabela 6: Reduções ao comparar-se o gasto energético total.

Conclusão

Sumário

Introdução

Fundamentação Teórica

MPPA-256
OpenMP API
Modelo mestre-escravo
MPPA IPC API
MPPA ASYNC API
CAP-Benchmarks

Desenvolvimento


Friendly Numbers
Fast
Gaussian Filter
LU Factorization
K-Means
Integer-Sort

Resultados

Desempenho
Tráfego de dados
Consumo energético

Conclusão

- ASYNC se sobressai em comparação a IPC
- Diretivas assíncronas são uma boa escolha em diversos contextos
- Comunicações assíncronas tem grande potencial de ganho em desempenho
- Trabalhos futuros
 - Comparar MPPA com outros processadores do estado da arte



Comparação de Tecnologias de Comunicação entre Clusters no Processador MPPA-256: Um Estudo com Aplicações do CAP Benchmarks

David Grunheidt Vilela Ordine, Márcio Castro

Departamento de Informática e Estatística (INE)
Universidade Federal de Santa Catarina (UFSC)
`david.ordine@grad.ufsc.br`
`marcio.castro@ufsc.br`

27 de dezembro de 2020

Referências

Sumário

Introdução

Fundamentação Teórica

MPPA-256
OpenMP API
Modelo mestre-escravo
MPPA IPC API
MPPA ASYNC API
CAP-Benchmarks

Desenvolvimento

Friendly Numbers
Fast
Gaussian Filter
LU Factorization
K-Means
Integer-Sort

Resultados

Desempenho
Tráfego de dados
Consumo energético

Conclusão



de Dinechin et al.

A Distributed Run-Time Environment for the Kalray MPPA-256 Integrated Manycore Processor.
In International Conference on Computational Science (ICCS), volume 18, pages 1654–1663, Barcelona, Spain, 2013. Elsevier.



H. Fu et al.

The sunway taihulight supercomputer: System and applications.
SCIENCE CHINA Information Sciences, 59(7):1–16, 2016.



Hascoët et al.

Asynchronous one-sided communications and synchronizations for a clustered manycore processor.
In Proceedings of the 15th IEEE/ACM Symp. on Embedded Systems for Real-Time Multimedia - ESTIMedia '17, pages 51–60, New York, New York, USA, 2017. ACM Press.



Olofsson et al.

Kickstarting high-performance energy-efficient manycore architectures with epiphany.
In Asilomar Conf. on Signals, Systems and Computers, pages 1719–1726. IEEE, nov 2014.



Penna et al.

An operating system service for remote memory accesses in low-power noc-based manycores.
In 12th IEEE/ACM International Symposium on Networks-on-Chip, Torino, Italy, 2018.



Souza et al.

CAP Bench: A Benchmark Suite for Performance and Energy Evaluation of Low-power Many-core Processors.
Concurrency and Computation: Practice and Experience, 29(4):e3892, 2016.