Parallel Computing



Master Informatics Eng.

2021/22 *A.J.Proença*

Background to this course (slides from several sources)

AJProença, Parallel Computing, MEI, UMinho, 2021/22

The course website:

http://gec.di.uminho.pt/mei/cp/



Direitos de Autor & Copyright

Avisos | Equipa docente | Objetivos e Organização | Resultados de Aprendizagem | Programa | Bibliografia | Sumários | Avaliação

2020/2021

Ultima Modificação: 30 Set 2021

| departamento de informática | | | \sim |
|---|-----------------------|--|--------|
| Equipa docente | Notas complementares: | | |
| Docentes responsáveis pela lecionação da UC | • | UC para MIEI, MEI & LEF UC de JLS, excecionalmente com AJP este ano <i>background</i> esperado: Lic Eng ^a Informática | |
| Teóricas Alberto José Proença <i>email</i> : aproenca <at>di.uminho.pt</at> | • | 1ªs semanas para revisão de conceitos básicos língua de comunicação escrita: inglês data última modificação do <i>website</i> : confiável | |
| Práticas Laboratoriais André Martins Pereira <i>email</i> : ampereira <at>di.uminho.pt Rui Silva <i>email</i>: ruisilva<at>di.uminho.pt</at></at> | • • • | convém ler sempre os avisos sumários e slides: atualizados semanalmente feriados em dezembro turnos práticos: possível usar horário de PLs em LEF | |

Courses timetable

 \sim

| | | | | 1 | 1 | 1 | |
|-------|--------------------------|---------------------------|-------------------------|-------------------------|-------------------------|---------------------------|-------|
| | 2-Segunda | 3-Terça | 4-Quarta | 5-Quinta | | 6-Sexta | |
| 08:30 | | | | | | | 08:30 |
| 09:00 | | | Computação Paralela MEI | Computação Paralela MEI | Computação Paralela MEI | | 09:00 |
| 09:30 | | | PL4 | PL1 | PL3 | | 09:30 |
| 10:00 | | | Rui Silva | André Pereira | Rui Silva | | 10:00 |
| 10:30 | | | DI 1.10 | DI 0.09 | DI 0.11 | | 10:30 |
| 11:00 | Programação Paralela LEF | | Computação Paralela MEI | | | Programação Paralela I FF | 11:00 |
| 11:30 | т1 | | PL5 | | | PI 1 | 11:30 |
| 12:00 | Alberto Proença | | Rui Silva | | | André Pereira | 12:00 |
| 12:30 | CP2 1.11 | | DI 1.10 | | | DI 0.04 | 12:30 |
| 13:00 | :00 ALMOÇO | | Computação Paralela MEI | | | 13:00 | |
| 13:30 | | | | CP1 | 0.08 | | 13:30 |
| 14:00 | | Ducence of a Develop 155 | | Computação Paralela MEI | Computação Paralela MEI | | 14:00 |
| 14:30 | | ri ogramaçao Paraleia LEF | | PL2 | PL6 | | 14:30 |
| 15:00 | | André Pereira | | André Pereira | Rui Silva | | 15:00 |
| 15:30 | | DI 0.04 | | DI 0.09 | DI 0.11 | | 15:30 |
| 16:00 | | | | | | | 16:00 |

AJProença, Parallel Computing, MEI, UMinho, 2021/22

Focus on this course: performance engineering

\sim

How:

- understanding the organization of computer system (its <u>architecture</u>) to develop efficient <u>algorithms</u> and <u>program</u> & <u>data structures</u>
- profiling & measuring the execution efficiency

Where in the hardware:

- in a memory hierarchy
 - multi-level caches
- in sequential code with ILP
 - pipelining
 - superscalar w/ out-of-order exec
 - vector processing
- in code with thread parallelism
 - multithreading in-core (SMT)
 - multithreading in multicore
 - multithreading in multiple devices
- in code with process parallelism
 - multiprocessing in a computer cluster

AJProença, Parallel Computing, MEI, UMinho, 2021/22



Computação Paralela

Níveis de paralelismo

- Instrução (ILP)
 - Execução de múltiplas instruções de um programa em paralelo

Core 0

L2

Shanghai

Core 2

L2

Core 3

L2

Core 1

L2

Shanghai

ared L3 Cache (non-inclusiv

Core 5

Core 4

L2

ad

Core 6

L1

Core 7

L2

IMC

- Processamento vetoria
- Limitado pelas deper do programa
- Tarefas / fios de execu
 - múltiplos fluxos instantes de una executam en programa executam ex
 - Limitado pelas dependências e características do algoritmo
- Processos
 - Múltiplos processos de um mesmo programa / ou vários programas





BACKGROUND AND TEST SYSTEMS

Dual-socket SMP systems based on AMD Opteron 23** (Shanghai) and Intel Xeon 55** (Nehalem-EP) processors have a similar high level design as depicted in Figure 1. The L1 and L2 caches are implemented per core, while the L3 cache is shared among all cores of one processor. The serial point-to-point links *HumerTransmort* (HT) and *Quick* AMD's last lev clusive nor inclus L3 cache into the the L3. Accordir (further details a by one core, otha processors use ex

Computação Paralela

Memória partilhada vs. memória distribuída

Memória partilhada

- Processadores partilham um canal de acesso à memória
- Caches reduzem o tráfego e a latencia dos acessos à memória
- Largura de banda de acesso à memória partilhada por várias UPs => limitação à escalabilidade deste tipo de arguitetura

Memória distribuída

Cada processador contém a sua própria memória, existindo uma rede de interligação entre os processadores

Sistemas híbridos

Acesso não uniforme à memória (NUMA)



Intel® Xeon® Scalable Processor Family Architected for the Data Center

- 2 socket servers have 2 CPUs 2 NUMA domains
- Difficult to get optimal application performance

• 2 socket server has 8 different desktop dies





Core 0

Core 1

Figure 1: Block diagram of the AMD (left) and Intel



bits within the L3 cache indicate that a cache line may be present in a certain core. If a bit is not set, the associated core certainly does not hold a copy of the cache line, thus reducing snoop traffic to that core. However, unmodified cache lines may be evicted from a core's cache without notification of the L3 cache. Therefore, a set core valid bit does not guarantee a cache line's presence in a higher level cache.

AMD's last clusive nor in L3 cache into the L3. Accor (further detai by one core. processors 11s protocol to er sors implement state owned write-back to the MESIF pr sure that shar The configu

ble 1. The list the main mer halem's three two DDR2-66 CPU but not ory performan ing to new so We disable

troduces resu Our benchma all caches and tageous SMT for all latency tions that dist bandwidth me hardware pref



- · Consistent performance, consistent latency

8 different NUMA domains

Repurposed Desktop Die for Server

4

AMD Naples



Computação Paralela

Os alunos deverão ser capazes de projetar e **otimizar** aplicações que executem de forma **eficiente** nas arquiteturas (paralelas) atuais:

- Escrever códigos sequenciais eficientes
- Implementar e otimizar códigos paralelos em sistemas de memória partilhada
- Desenhar e implementar códigos para sistemas de memória distribuída

Programa (versão resumida)

Desempenho "single core"

- Medição de desempenho e métricas
- Hierarquia de memória
- Otimizações para paralelismo ao nível da instrução

Programação de sistemas de memória partilhada

- Modelos baseados em fios de execução
- Medição e otimização de desempenho

Sistemas de memória distribuída

- Modelo de processos comunicantes
- Desenho de aplicações

Algoritmos paralelos

Análise da complexidade e escalabilidade

Background for Advanced Architectures



Key concepts to revise:

- numerical data representation (integers & FP)
- ISA (Instruction Set Architecture)
- how C compilers generate code (a look into assembly code)
 - how scalar and structured data are allocated
 - how control structures are implemented
 - how to call/return from function/procedures
 - what architecture features impact performance
- improvements to enhance performance in a single PU
 - ILP: pipeline, multiple issue, ...
 - data parallelism: SIMD/vector processing, ...
 - thread-level parallelism
 - memory hierarchy: cache levels, ...

Keyword: paralellism

Key textbook (1)



Table of Contents

Printed Text

- 1. Fundamentals of Quantitative Design and Analysis
- 2. Memory Hierarchy Design
- 3. Instruction-Level Parallelism and Its Exploitation
- 4. Data-Level Parallelism in Vector, SIMD, and GPU Architectures
- 5. Multiprocessors and Thread-Level Parallelism
- 6. The Warehouse-Scale Computer
- 7. Domain Specific Architectures
- A. Instruction Set Principles
- B. Review of Memory Hierarchy
- C. Pipelining: Basic and Intermediate Concepts

Online

- D. Storage Systems
- E. Embedded Systems
- F. Interconnection Networks
- G. Vector Processors
- H. Hardware and Software for VLIW and EPIC
- I. Large-Scale Multiprocessors and Scientific Applications
- J. Computer Arithmetic
- K. Survey of Instruction Set Architectures
- L. Advanced Concepts on Address Translation
- M. Historical Perspectives and References

A New Golden Age for Computer Architecture

Growth of computer performance

BY JOHN L. HENNESSY AND DAVID A. PATTERSON









PARALLEL COMPUTING ARCHITECTURES AND APIS

IoT Big Data Stream Processing

Vivek Kale



TABLE OF CONTENTS

- 1. Uniprocessor Computers
- 2. Processor Physics and Moore's Law
- Section I Genesis of Parallel Computing
 - 3. Processor Basics
 - 4. Networking Basics
 - 5. Distributed Systems Basics

Section II Road to Parallel Computing

- 6. Parallel Systems
- 7. Parallel Computing Models
- 8. Parallel Algorithms

Section III Parallel Computing Architectures

- 9. Parallel Computing Architecture Basics
- 10. Shared Memory Architecture
- 11. Message-Passing Architecture
- 12. Stream Processing Architecture

Section IV Parallel Computing Programming

- 13. Parallel Computing Programming Basics
- 14. Shared-memory Parallel Programming with OpenMP
- 15.Message Passing Parallel Programming with MPI
- 16. Stream Processing Programming with CUDA, OpenCL, and OpenACC

Section V Internet of Things Big Data Stream Processing

- 17. Internet of Things (IoT) Technologies
- 18. Sensor Data Processing
- 19. Big Data Computing
- 20. Big Data Stream Processing

Epilogue: Quantum Computing

Key textbook (3)

Structured Parallel Programming

Patterns for Efficient Computation

Michael McCool, Arch D. Robison, James Reinder

M<

Table of contents

- 1 Introduction
- 2 Background
- 3 Patterns
- 4 Map
- 5 Collectives
- 6 Data Reorganization
- 7 Stencil and Recurrence
- 8 Fork-Join
- 9 Pipeline
- 10 Forward Seismic Simulation
- 11 K-Means Clustering
- 12 Bzip2 Data Compression
- 13 Merge Sort
- 14 Sample Sort
- 15 Cholesky Factorization Appendices

Recommended textbook (1)



Contents

1. Introduction

- 2. Data parallel computing
- 3. Scalable parallel execution
- 4. Memory and data locality
- 5. Performance considerations
- 6. Numerical considerations
- 7. Parallel patterns: Convolution
- 8. Parallel patterns: Prefix Sum
- 9. Parallel patterns : Parallel Histogram Computation
- 10. Parallel patterns: Sparse Matrix Computation
- 11. Parallel patterns: Merge Sort
- 12. Parallel patterns: Graph Searches
- 13. CUDA dynamic parallelism
- 14. Application case study-non-Cartesian magnetic ...
- 15. Application case study—molecular visualization ...
- 16. Application case study—machine learning
- 17. Parallel programming and computational thinking
- 18. Programming a heterogeneous computing cluster
- 19. Parallel programming with OpenACC
- 20. More on CUDA and graphics processing computing
- 21. Conclusion and outlook

Appendix A. An introduction to OpenCL Appendix B. THRUST: a productivity-oriented library for CUDA

Recommended textbook (2)

INTEL® XEON PHIM PROCESSOR HIGH PERFORMANCE PROGRAMMING KNIGHTS LANDING EDITION

Jim Jeffers | James Reinders | Avinash Sodani



Table of Contents

Section I: Knights Landing.

Chapter 1: Introduction Chapter 2: Knights Landing Overview Chapter 3: Programming MCDRAM and Cluster Modes Chapter 4: Knights Landing Architecture Chapter 5: Intel Omni-Path Fabric Chapter 6: uarch Optimization Advice

Section II: Parallel Programming

Chapter 7: Programming Overview for Knights Landing Chapter 8: Tasks and Threads Chapter 9: Vectorization Chapter 10: Vectorization Advisor Chapter 11: Vectorization with SDLT Chapter 12: Vectorization with AVX-512 Intrinsics Chapter 13: Performance Libraries Chapter 14: Profiling and Timing Chapter 15: MPI Chapter 15: MPI Chapter 16: PGAS Programming Models Chapter 17: Software Defined Visualization Chapter 18: Offload to Knights Landing Chapter 19: Power Analysis

Section III: Pearls

Chapters 20-26: Results on LAMMPS, SeisSol, WRF, N-Body Simulations, Machine Learning, Trinity mini-applications and QCD are discussed.

Background concepts from a basic Computer Systems course

Some notes/comments for this course:

- some slides are borrowed from

公



AJProença, Parallel Programming, LEF, UMinho, 2021/22

Desempenho do CPU

Previsão do tempo de execução (T_{EXEC}) de um programa numa máquina - requer um **modelo** que relacione o desempenho com as características do sistema de computação (*hw*+*sw*)

Um programa numa máquina executa num determinado número <u>médio</u> de ciclos de relógio: **# clock cycles**

O período do relógio do CPU é constante: **Tcc = 1 / f**

T_{EXEC} = # clock cycles * Tcc

Desempenho do CPU

• De que depende o número médio de ciclos necessários para executar um programa?



The BIG Picture



AJProença, Parallel Computing, MEI, UMinho, 2021/22

Desempenho do CPU

• Calcule o tempo de execução do programa abaixo numa máquina com um relógio de 2 GHz e CPI=1.5

#I = 32NOTA: número de instruções executadas.

$$T_{exec} = 32 * 1.5 / 2 E9 = 24 E-9 s = 24 ns$$

Relação entre as métricas

$T_{EXEC} = CPI^* \# I/f$

- #I depende do algoritmo, do compilador e da arquitectura (ISA)
- CPI depende da arquitectura (ISA), da mistura de instruções efectivamente utilizadas, da organização do processador e da organização dos restantes componentes do sistema (ex., memória)
- *f* depende da organização do processador e da tecnologia utilizada
- "A única métrica completa e fiável para avaliar o desempenho de um computador é o tempo de execução"
- As métricas CPI, *f* e #I não podem ser avaliadas isoladamente, devendo ser sempre consideradas em conjunto, pois dependem umas das outras.

Desempenho do CPU - MIPS

MIPS (milhões de instruções por segundo) – uma métrica enganadora

$$MIPS = \frac{\#I}{T_{exec} * 10^6}$$

- 1. MIPS especifica a taxa de execução das instruções, mas não considera o trabalho feito por cada instrução. CPUs com diferentes *instruction sets* não podem ser comparados.
- 2. MIPS varia entre diferentes programas no mesmo CPU
- 3. MIPS pode variar inversamente com o desempenho

Esta métrica pode ser usada para comparar o desempenho do mesmo programa em CPUs com o mesmo conjunto de instruções, mas micro-arquitecturas e/ou frequências do relógio diferentes.

Desempenho do CPU- MIPS

MIPS de pico-máxima taxa possível de execução de instruções

É a métrica mais enganadora, pois corresponde a sequências de código que apenas tenham instruções com o CPI mais baixo possível. Este tipo de sequências de instruções não realizam, regra geral, trabalho útil; consistem apenas em operações elementares com operandos em registos.

Pode ser visto como "a velocidade da luz" do CPU, e portanto, inatingível.

O principal problema é que é muitas vezes publicitada pelos fabricantes/vendedores como uma medida de desempenho das suas máquinas!

Background for Advanced Architectures



Key concepts to revise:

- numerical data representation (integers & FP)
- ISA (Instruction Set Architecture)
- how C compilers generate code (a look into assembly code)
 - how scalar and structured data are allocated
 - how control structures are implemented
 - how to call/return from function/procedures
 - what architecture features impact performance
- improvements to enhance performance in a single PU
 - ILP: pipeline, multiple issue, ...
 - data parallelism: SIMD/vector processing, ...
 - thread-level parallelism
 - memory hierarchy: cache levels, ...

Keyword: paralellism

Paralelismo no processador Exemplo 1

2

S1 S2 S3 **S**4 **S**5 Exemplo de Instruction Write Instruction Operand Instruction fetch decode fetch execution back pipeline unit unit unit unit unit (a) **Objetivo** S1: 2 4 5 8 S2: 3 7 1 6 •CPI = 1 7 5 6 3 2 4 S3: 2 3 5 6 4 S4: **Problemas**: S5: dependências de dados 2 3 8 1 4 9

Time

(b)

latências nos acessos à memória

•saltos condicionais; propostas de solução para minimizar perdas:

- executar sempre a instrução "que se segue"
- usar o historial dos saltos anteriores (1 ou mais bits)
- executar os 2 percursos alternativos até à tomada de decisão

AJProença, Sistemas de Computação, UMinho, 2020/21

Pipeline Summary

The BIG Picture

公

- Pipelining improves performance by increasing instruction throughput
 - Executes multiple instructions in parallel
 - Each instruction has the same latency
- Subject to hazards
 - Structure, data, control
- Instruction set design affects complexity of pipeline implementation

Paralelismo no processador Exemplo 2

2

Exemplo de superescalaridade (2 vias)



Does Multiple Issue Work?

The BIG Picture

公

- Yes, but not as much as we'd like
- Programs have real dependencies that limit ILP
- Some dependencies are hard to eliminate
 - e.g., pointer aliasing
- Some parallelism is hard to expose
 - Limited window size during instruction issue
- Memory delays and limited bandwidth
 - Hard to keep pipelines full
- Speculation can help if done well

AJProença, Parallel Computing, MEI, UMinho, 2021/22

27

Organização hierárquica da memória



AJProença, Sistemas de Computação, UMinho, 2020/21

A cache em arquiteturas multicore



Notas:

2

•as caches L1 de <u>dados</u> e de <u>instruções</u> são normalmente distintas
•as caches L2 em *multicores* podem ser partilhadas por outras cores
•muitos cores partilhando uma única memória traz complexidades:

- manutenção da coerência da informação nas caches
- encaminhamento e partilha dos circuitos de acesso à memória

Multicore architecture in a server system





The cluster at DI



http://search6.di.uminho.pt/wordpress/?page_id=274

Multicore architectures

\sim

Questions/homework:

- 1. Identify the current available devices with the largest #cores; state how many in the device/package & show an image
 - a) Designed by Intel
 - b) Designed by AMD
 - c) Designed by ARM
 - d) Designed by a japanese company
 - e) Designed by chinese company
 - f) Worldwide
- 2. What are the key chalenges to design a chip with a very large number of cores?

