Parallel Programming
Josep M. Codina
Intel Barcelona Research Center

Aula Empresa, Facultat d’Informàtica de Barcelona, February 2010
© Intel Corporation, 2010

Why Parallel Programming?
• Applications are naturally parallel
• Multi-core are out there
• We need to think in parallel!!!!

Agenda
• Motivation
• Designing Parallel Applications
• Parallel Programming Models
• Automatic Parallelization
• Parallel Programming for Games
• Concluding Remarks

Design of Parallel Applications
• Understand the Problem
  – Identify hotspots
    • Profile application
  – Identify bottlenecks to the parallelization
  – Communication
  – Synchronization
  – Consider alternative versions of the application
• Split Application in Parts
  – Maximum parallelism
  – Minimum imbalance
  – Minimum waiting time

Who Creates Parallel Applications?
• User/ Programmer
  – Parallel languages and libraries
• Compilers
  – Automatic Parallelization
• Parallel Architectures
  – Multi-core in a single chip
  – Many multi-core chips

Parallel Programming?
I still want to divide the effort of eating pizza to eat all slices as fast as possible!!

But, I wonder...
How we split this hard task 😊

48-10 cores in the Intel “Single-chip Cloud Computer”
What Do We Need?

Parallel Programming Languages

Provide Tools Today

Research Tomorrow’s Techniques

Educate Tomorrow’s Experts

When to Parallelize an Application?

- Always!!
- Stop thinking in sequential applications
- Multicore era is here and we have to leverage the computing capabilities that we have out there

Agenda

- Motivation
- Designing Parallel Applications
- Parallel Programming Models
- Automatic Parallelization
- Parallel Programming for Games
- Concluding Remarks

Parallel Programming Models

- Data Parallel
- Message Passing
- Shared Memory
- Distributed Shared Memory Model

NOTE: These models are orthogonal to the actual hardware!!

Data Parallel Model

- Set of cooperating tasks on the same data structure
- Tasks perform the same operation over different data items

Advantage
- Easy to write and comprehend
- No synchronization
- No independent code

Example: HPF (High Performance Fortran)

Message Passing Model

- A set of cooperating sequential processes
- Each with own local address space
- Processes interact with explicit transaction (send, receive,...)

Advantage
- Programmer controls data and work distribution

Disadvantage
- Communication overhead for small transactions
- Hard to program!

Example: MPI

NOTE: These models are orthogonal to the actual hardware!!

Designing Tomorrow’s Microprocessors

Educate Tomorrow’s Experts

- Helped 45 universities add parallel programming courses
- 7500 students took them
- 2007 Goal: 400+ universities

Research Tomorrow’s Techniques

- Transactional Memory
- Speculative Multi-threading
- Data Parallel Languages
Shared Memory Model

- Different simultaneous execution threads (processes)
  - Read / Write to one shared memory space and invalidate if necessary

Advantage
- Read remote memory via an expression
- Write remote memory through assignment

Disadvantage
- Manipulating shared data leads to synchronization requirements
- Does not allow locality exploitation

Example: OpenMP

Distributed Shared Memory Model

- Similar to the shared memory paradigm
  - Memory M has affinity to Thread i

Advantage
- Helps exploiting locality of references
- Simple statements as SM

Disadvantage
- Synchronization still necessary
- Example: UPC, Titanium, Co-Array, Global Arrays

Agenda

- Motivation
- Designing Parallel Applications
- Parallel Programming Languages
- Automatic Parallelization
- Parallel Programming for Games
- Concluding Remarks

Traditional Auto Parallelization

- Automatic decomposition of applications into threads
- No need for inserting directives or pragmas
- Compiler identifies suitable parts of the application for parallelization
  - Typically simple loops
  - Considering simple memory disambiguation schemes
- This approach is typically limited to simple applications
  - Dependences limit its applicability to large scale applications

Speculative Multithreading

- Automatic decomposition of applications into speculative threads
  - No need to be conservative on the memory disambiguation

Agenda

- Motivation
- Designing Parallel Applications
- Parallel Programming Languages
- Automatic Parallelization
- Parallel Programming for Games
- Concluding Remarks
Usual Game Structure

- Physics – AI – Particles

Designing Tomorrow’s Microprocessors

Comparative Analysis of Game Parallelization, Dimitry Eremin, GDC’08

Game Thread Profiling

Sequential execution
25% of system utilization
Benchmark: 20.95sec
Measured on a 4 core test machine

OnFrameMove

Render

Physics

AI

Particles

Game Thread Profiling

OnFrameMove

Render

Physics

AI

Particles

Designing Tomorrow’s Microprocessors

Comparative Analysis of Game Parallelization, Dimitry Eremin, GDC’08

Functional Decomposition

• Load imbalance
• Benchmark: 10.15sec

Concurrency level is the number of threads (core is a number that is a factor, generally, listener, etc.) at a given time.

• Load imbalance
• Benchmark: 9.44 sec

Functional and Data Decomposition

• If Split AI in 2 threads
• Load imbalance
• Benchmark: 9.44sec

Intel® Threading Building Blocks

• Open Source Library for parallelism in C++
  – Fine-grain decomposition
  – Task scheduler

Good utilization of 4 cores
Benchmark: 8.66 sec

Comparative Analysis of Game Parallelization, Dimitry Eremin, GDC’08
Agenda

• Motivation
• Designing Parallel Applications
• Parallel Programming Languages
• Automatic Parallelization
• Parallel Programming for Games
• Concluding Remarks

Summary - Conclusions

• Parallel Applications are required to leverage Parallel Architectures
• Today we discussed about:
  – Why we need parallel applications
  – Considerations When Designing parallel applications
  – Parallel Programming Models
  – Auto Parallelization and Speculative Multithreading
  – The importance of threading in gaming
• We need to think in parallel to create parallel applications!!!