A “Hands-on” Introduction to OpenMP

Tim Mattson
Intel Corp.
timothy.g.mattson@intel.com

* The name “OpenMP” is the property of the OpenMP Architecture Review Board.
Introduction

- OpenMP is one of the most common parallel programming models in use today.
- It is relatively easy to use which makes a great language to start with when learning to write parallel software.

Assumptions:
- We assume you know C. OpenMP supports Fortran and C++, but we will restrict ourselves to C.
- We assume you are new to parallel programming.
- We assume you have access to a compiler that supports OpenMP (more on that later).
Acknowledgements

- This course is based on a long series of tutorials presented at Supercomputing conferences. The following people helped prepare this content:
  - J. Mark Bull (the University of Edinburgh)
  - Rudi Eigenmann (Purdue University)
  - Barbara Chapman (University of Houston)
  - Larry Meadows, Sanjiv Shah, and Clay Breshears (Intel Corp).

- Some slides are based on a course I teach with Kurt Keutzer of UC Berkeley. The course is called “CS194: Architecting parallel applications with design patterns”. These slides are marked with the UC Berkeley ParLab logo:
Preliminaries:

- Our plan ... Active learning!
  - We will mix short lectures with short exercises.
- Download exercises and reference materials.
- Please follow these simple rules
  - Do the exercises we assign and then change things around and experiment.
  - Embrace active learning!
  - Don’t cheat: Do Not look at the solutions before you complete an exercise ... even if you get really frustrated.
Outline

- **Unit 1: Getting started with OpenMP**
  - Mod 1: Introduction to parallel programming
  - Mod 2: The boring bits: Using an OpenMP compiler (hello world)
  - Disc 1: Hello world and how threads work
- **Unit 2: The core features of OpenMP**
  - Mod 3: Creating Threads (the Pi program)
  - Disc 2: The simple Pi program and why it sucks
  - Mod 4: Synchronization (Pi program revisited)
  - Disc 3: Synchronization overhead and eliminating false sharing
  - Mod 5: Parallel Loops (making the Pi program simple)
  - Disc 4: Pi program wrap-up
- **Unit 3: Working with OpenMP**
  - Mod 6: Synchronize single masters and stuff
  - Mod 7: Data environment
  - Disc 5: Debugging OpenMP programs
  - Mod 8: Skills practice … linked lists and OpenMP
  - Disc 6: Different ways to traverse linked lists
- **Unit 4: a few advanced OpenMP topics**
  - Mod 8: Tasks (linked lists the easy way)
  - Disc 7: Understanding Tasks
  - Mod 8: The scary stuff … Memory model, atomics, and flush (pairwise synch).
  - Disc 8: The pitfalls of pairwise synchronization
  - Mod 9: Threadprivate Data and how to support libraries (Pi again)
  - Disc 9: Random number generators
- **Unit 5: Recapitulation**
Outline

- **Unit 1: Getting started with OpenMP**
  - Mod1: Introduction to parallel programming
  - Mod 2: The boring bits: Using an OpenMP compiler (hello world)
  - Disc 1: Hello world and how threads work

- **Unit 2: The core features of OpenMP**
  - Mod 3: Creating Threads (the Pi program)
  - Disc 2: The simple Pi program and why it sucks
  - Mod 4: Synchronization (Pi program revisited)
  - Disc 3: Synchronization overhead and eliminating false sharing
  - Mod 5: Parallel Loops (making the Pi program simple)
  - Disc 4: Pi program wrap-up

- **Unit 3: Working with OpenMP**
  - Mod 6: Synchronize single masters and stuff
  - Mod 7: Data environment
  - Disc 5: Debugging OpenMP programs
  - Mod 8: Skills practice … linked lists and OpenMP
  - Disc 6: Different ways to traverse linked lists

- **Unit 4: a few advanced OpenMP topics**
  - Mod 8: Tasks (linked lists the easy way)
  - Disc 7: Understanding Tasks
  - Mod 8: The scary stuff … Memory model, atomics, and flush (pairwise synch).
  - Disc 8: The pitfalls of pairwise synchronization
  - Mod 9: Threadprivate Data and how to support libraries (Pi again)
  - Disc 9: Random number generators

- **Unit 5: Recapitulation**
In 1965, Intel co-founder Gordon Moore predicted (from just 3 data points!) that semiconductor density would double every 18 months.

He was right! Transistors are still shrinking as he projected.
Consequences of Moore’s law...

The good old days ...

(SPECint)
Uniprocessor Performance


Third party names are the property of their owners.
The Hardware/Software contract

- Write your software as you choose and we HW-geniuses will take care of performance.

- The result: Generations of performance ignorant software engineers using performance-handicapped languages (such as Java) … which was OK since performance was a HW job.
... Computer architecture and the power wall

Scalar Performance

power = perf ^ 1.74

Pentium 4 (Psc)
Pentium 4 (Wmt)
Pentium Pro
Pentium

Growth in power is unsustainable

Source: E. Grochowski of Intel
... partial solution: simple low power cores

Eventually Pentium 4 used over 30 pipeline stages!!!
For the rest of the solution consider power in a chip ...

C = capacitance ... it measures the ability of a circuit to store energy:

\[ C = \frac{q}{V} \Rightarrow q = CV \]

Work is pushing something (charge or q) across a "distance" ... in electrostatic terms pushing q from 0 to V:

\[ V \times q = W. \]

But for a circuit \( q = CV \) so

\[ W = CV^2 \]

Power is work over time ... or how many times in a second we oscillate the circuit

\[ \text{Power} = W \times F \Rightarrow \text{Power} = CV^2f \]
... The rest of the solution
add cores

Capacitance = C
Voltage = V
Frequency = f
Power = CV²f

Capacitance = 2.2C
Voltage = 0.6V
Frequency = 0.5f
Power = 0.396CV²f

Chandrakasan, A.P.; Potkonjak, M.; Mehra, R.; Rabaey, J.; Brodersen, R.W.,
"Optimizing power using transformations," IEEE Transactions on Computer-Aided

Source:
Vishwani Agrawal
Microprocessor trends

Individual processors are many core (and often heterogeneous) processors.

Intel SCC Processor

48 cores

AMD ATI RV770

10 cores
16 wide SIMD

NVIDIA Tesla C1060

30 cores
8 wide SIMD

Intel® Xeon® processor

4 cores

IBM Cell

1 CPU + 6 cores

ARM MPCORE

4 cores

Source: OpenCL tutorial, Gaster, Howes, Mattson, and Lokhmovtov, HiPEAC 2011
The result…

A new contract … HW people will do what’s natural for them (lots of simple cores) and SW people will have to adapt (rewrite everything)

The problem is this was presented as an ultimatum … nobody asked us if we were OK with this new contract … which is kind of rude.
Concurrency vs. Parallelism

- Two important definitions:
  - **Concurrency**: A condition of a system in which multiple tasks are *logically* active at one time.
  - **Parallelism**: A condition of a system in which multiple tasks are *actually* active at one time.

Figure from “An Introduction to Concurrency in Programming Languages” by J. Sottile, Timothy G. Mattson, and Craig E Rasmussen, 2010
Concurrency vs. Parallelism

- Two important definitions:
  - **Concurrency**: A condition of a system in which multiple tasks are *logically* active at one time.
  - **Parallelism**: A condition of a system in which multiple tasks are *actually* active at one time.

Programs

Concurrent Programs

Parallel Programs

Figure from “An Introduction to Concurrency in Programming Languages” by J. Sottile, Timothy G. Mattson, and Craig E Rasmussen, 2010
Concurrent vs. Parallel applications

- We distinguish between two classes of applications that exploit the concurrency in a problem:

  - **Concurrent application**: An application for which computations **logically** execute simultaneously due to the semantics of the application.
    - The problem is fundamentally concurrent.

  - **Parallel application**: An application for which the computations **actually** execute simultaneously in order to complete a problem in less time.
    - The problem doesn’t inherently require concurrency … you can state it sequentially.
The Parallel programming process:

Original Problem

Tasks, shared and local data

Algorithm strategy

Implementation strategy

Units of execution + new shared data for extracted dependencies

Find Concurrency

Corresponding source code

Program SPMD_Emb_Par()
{
    TYPE *tmp, *func();
    global_array Data(TYPE);
    global_array Res(TYPE);
    int Num = get_num_procs();
    int id = get_proc_id();
    if (id==0) setup_problem(N, Data);
    for (int I=0; I<N; I=I+Num)
    {
        tmp = func(I, Data);
        Res.accumulate(tmp);
    }
}