ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

CSCS
Swiss National Supercomputing Centre

DAY 2: Historical Prospective on MPI and OpenMP Programming

Multi-threaded Programming, Tuning and Optimization on Multi-core MPP Platforms

15-17 February 2011

CSCS, Manno

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

CSCS
Swiss National Supercomputing Centre

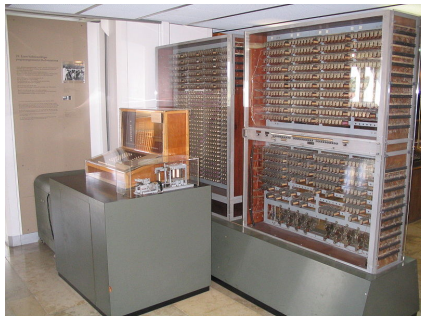
MPI/OpenMP Hybrid Parallelism for Multi-core Processors

Purpose of this course

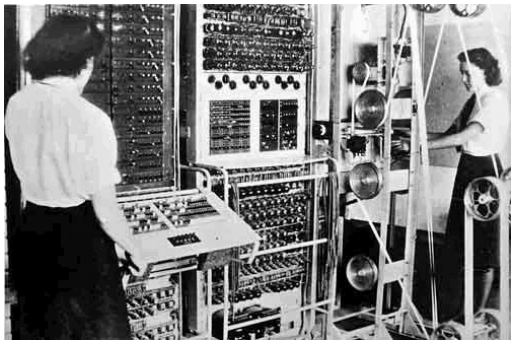
- This course concentrates on multi-threaded programming on Cray XE6 systems
- The purpose is to give an overview of multi-threaded programming for people who are familiar with MPI programming and are thinking of producing MPI/ OpenMP hybrid code
- … to think about why you might want to take this approach
  - Are you likely to gain anything by taking this approach ?
  - Is it likely to be portable and future-proof
- … to get you to think about your application before doing any coding
- … to give you a brief understanding of multi-socket multi-core node architectures
- … to give you some basic information about the operating system (CNL, a flavour of Linux)
- … to point out some potential problems that you might encounter
- … to describe the performance issues that you need to be aware of
- … to show you that there are tools available to help you in your task
- … to describe the threaded libraries provided on Cray XE6 machines

Historical Overview

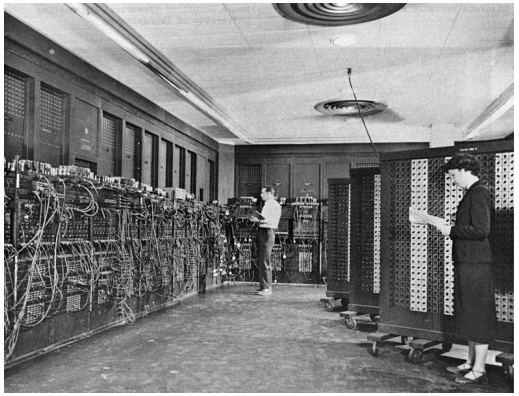
… The First Digital Computers …



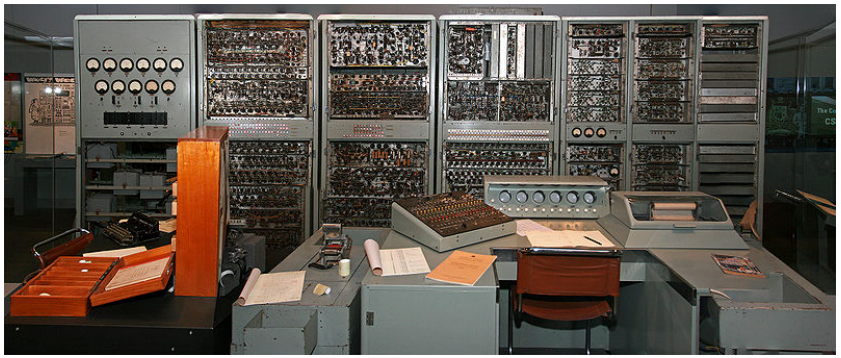
Zuse Z3 (Germany)



Colossus (UK)



ENIAC (USA)



CSIRAC (Australia)

… Some Programming Languages …

- Fortran was the first high-level programming language
  - … and it still appears to be the most common in HPC
- The first Fortran compiler was created in 1957
- … a set of other programming languages which are still in use such as Lisp …
- Then followed a number of other programming languages such as Algol, Pascal and a language called BCPL which gave rise to …
- … C which was created in 1973 in Bell labs, and which in turn led to …
- … C++ which was created as “C with classes” in 1980

- We are using programming languages that have been in use for 30, 40 or even 50 years
- *Are our parallel programming models likely to be as robust ?*

… Parallelism Enters the Game …

- People were thinking about parallelism in the 1960s
  - Amdahl's law was stated in 1967
    - If P is the part of a program that can be parallelised, and this part can be sped up N times, then the overall speedup of the code is given by

$$Speedup = \frac{1}{(1-P) + \frac{P}{N}}$$

- The programming language Algol-68 was designed with a number of constructs to help in concurrent programming and parallelism
- The supercomputer world began to use vector computers which were helped by the use of vectorising compilers
- Parallelism appeared in languages such as ILTRAN, Tranquil and Algol-68
- OCCAM was devised for parallel transputers

… Parallel Supercomputers Arrived …

- Supercomputers
  - CDC-6600 is often credited as being the first supercomputer in 1964
    - It could operate at over 10 Mflop/s
      - That's similar power to an iPhone
- The next generation were Parallel Vector Computers
  - ILLIAC IV, Cray-1
- By the 1990s we were moving towards the era of Massively Parallel Processing machines
  - Intel Paragon, Cray T3D/T3E
- The MPPs were complemented by large shared memory machines such as the SGI Origin series

… Programming Models Evolve for Parallelism …

- PVM (parallel virtual machine) was developed as the first widely-used **message passing** model and library
- **Compiler directives** had been in use to help vectorisation and now spread …
- … compiler directives were used for shared memory parallelism on a variety of machines
- Compiler directives also made their way in to distributed memory programming with High Performance Fortran (mid 1990s)
  - Poor initial performance led most people to abandon HPF as soon as it was created
  - Experience on Earth Simulator showed that it could be a successful programming paradigm

… Distributed Memory and Shared Memory …

- Ultimately, distributed memory programming was too complex for a *compiler* to convert a standard sequential source code into a distributed memory version
  - Although HPF tried to do this with directives
    - XcalableMP is a similar modern attempt to produce a HPF-like set of directives
- Programming a distributed memory computer was therefore to be achieved through message passing
  - Explicit messages introduced by the *code developer*, **not** by a compiler
- With the right kind of architecture it was possible to convert a sequential code into a parallel code that could be run on a shared memory machine
  - Although for performance you should "think parallel" when designing your application
- Programming for shared memory machines developed to use compiler directives
- For shared memory programming, each vendor produced their own set of directives
  - Not portable
  - Similar to the multitude of compiler directives in existence today for specific compiler optimisations (IVDEP etc.)

… Standardisation of Programming Models …

- Finally … the distributed memory message passing model settled on MPI as the *de facto* standard
- Shared memory programming centred around a standard developed by the OpenMP forum
- **Both have now been around for about 15 years**
- **These have been the dominant models for over 10 years**
- Other programming languages and programming models may take over in due time, but MPI and OpenMP will be around for a while yet …

Why MPI/OpenMP ?

The Need for MPI on Distributed Memory Clusters

- Large problems need lots of memory and processing power which is only available on distributed memory machines
- In order to combine this memory and processing power to be used by a single tightly-coupled application we need to have a programming model which can unify these separate processing elements
  - This was the reason that message passing models with libraries such as PVM evolved
- Currently the most ubiquitous programming model is message passing, and the most widely used method of carrying this out is using MPI
- MPI can be used for most of the explicit parallelism on modern HPC systems

- But what about the programming model within an individual node of a distributed memory machine …

Historical usage of MPI/OpenMP

- MPI-OpenMP hybrid programming is not new
- It was used to parallelise some codes on the early IBM pseries p690 systems
  - These machines were shipped with a quite weak “colony” interconnect
  - CSCS had a 256 processor IBM p690 system between 2002 and 2007
- There are a number of codes are in use today that benefitted from this work
  - The most prominent example is CPMD
- Several publications were made
  - E.g. *Mixed-mode parallelization (OpenMP-MPI) of the gyrokinetic toroidal code GTC to study microturbulence in fusion plasmas*
    - Ethier, Stephane; Lin, Zhihong
  - Recently the GTC developers have revisited MPI/OpenMP hybrid programming making use of the new TASK model

Simple cases where MPI/OpenMP might help

- If you have only partially parallelised your algorithm using MPI
  - E.g. 3D Grid-based problem using domain decomposition only in 2D
    - For strong scaling this leads to long thin domains for each MPI process
    - The same number of nodes using MPI/OpenMP hybrid might scale better
  - If there are a hierarchy of elements in your problem and you have only parallelised over the outer ones
- Where your problem size is limited by memory per process on a node
  - For example you might currently have to leave some cores idle on a node in order to increase the memory available per process
- Where compute time is dominated by halo exchanges or similar communication mechanism
- When your limiting factor is scalability and you wish to reduce your time to solution by running on more processors
  - In many cases you won't be faster than an MPI application whilst it is still in its scaling range
  - You might be able to scale more than with the MPI application though …

… in these cases and more, we can take advantage of the changes in architecture of the nodes that make up our modern HPC clusters …

Alternatives to MPI that you might wish to consider

- PGAS – partitioned global address space languages
  - These languages attempt to unify two concepts
    - the simplicity of allowing data to be globally visible from any processor
      - Ease of use and productivity enhancement
    - explicitly showing in the code where data is on a remote processor
      - Making it clear to the programmer where expensive “communication” operations take place
  - UPC [Unified Parallel C] provides a global view of some C data structures
  - Co-array Fortran
    - This has now been voted into the official Fortran standard
- Global Arrays toolkit
  - This is a popular API and library for taking a global view of some data structures
  - It is particularly popular in the computational chemistry community
    - NWChem is written on top of the GA toolkit

So why are we talking here about MPI ?

- MPI is ubiquitous
  - Available on clusters everywhere
    - Version provided with Linux
    - Vendor tuned libraries provided for high-end systems
    - Open-source libraries available (OpenMP, MPICH)
- Most codes are already written using MPI
  - Familiar interface for most people involved in computational science
- Has allowed scalability up to hundreds of thousands of cores
- Is still the *de facto* standard distributed programming model
- Is where most of the effort into improving distributed memory parallel programming models/interfaces is going

What MPI Provides

- The MPI standard defines an Application Programmer Interface to a set of library functions for message passing communication
- From MPI-2 a job launch mechanism `mpiexec` is described, but this is a suggestion and is not normally used
  - Most implementations use `mpirun` or some specialised job launcher
    - E.g. `srun` for the Slurm batch system, `aprun` for Cray, `poe` for IBM

… but the MPI standard does not specify …

- The method of compiling and linking an MPI application
  - Typically this might be `mpicc`, `mpif90`, `mpiCC` etc. for MPICH and OpenMPI implementations, but this is not portable
    - E.g. `cc/ftn` for Cray, `mpxlc/mpxlf90` for IBM
- The environment variables to control runtime execution
- Details of the implementation
  - It does provide *advice* to implementors and users

It is only the API that provides portability.

Alternatives to OpenMP that you might consider …

- Combining your MPI with Pthreads or other threading models
- Using Intel Thread Building Blocks on a node
  - C++ only
- Using a low level model for kernels such as OpenCL
  - OpenCL can target standard CPUs as well as GPUs and other accelerators
- Using Intel Array Building Blocks
  - Current version 1.0Beta3
  - Grew out of a combination of earlier products/efforts
    - Intel Ct
    - Rapidmind
    - Cilk
  - C++ only

PThreads

- An alternative model for combining threads with message passing is to use the POSIX Threads interface and library Pthreads
- Most OpenMP implementations are built on top of Pthreads !!
- Pthreads provides the ability to dynamically create threads which are launched to run a specific task
- Pthreads provides finer grained control than OpenMP
- The disadvantages of Pthreads compared to OpenMP are
  - There is no Fortran interface in the standard
    - Although IBM did produce a Fortran interface for their XLF compiler
  - You have to manage (re-create) any worksharing yourself
  - It is a low level interface
    - As MPI is often referred to as a low level interface this might not be a problem
  - You will typically take many more lines of coding using Pthreads than OpenMP
    - OpenMP is normally a more *productive* way of coding for numerical codes

Intel Thread Building Blocks

- Intel Thread Building Blocks (TBB) is a C++ template library that adds parallel programming for C++ programmers.
  - Not applicable to Fortran or C codes
- "Extends" C++ by adding a set of parallel keywords
  - These "extensions" are not actually real changes to the languages
  - As a template library, any standard-conforming C++ compiler can use TBB
- Algorithms can be parallelised in a number of ways
  - `parallel_for`, `parallel_reduce`, `parallel_while` etc.
- TBB adds containers, locks, a task scheduler etc.
- TBB was built out of positive user experiences from OpenMP, and the desire to provide an object-oriented, template based approach to parallelism in C++

So why are we talking here about OpenMP ?

- OpenMP is ubiquitous
  - Available with almost all compilers
    - gcc provides OpenMP support
    - Vendor tuned implementations available
- Easy interface available for incremental parallelism
- Is the best fit for data-parallel codes
- Is being updated to incorporate methods for modern programming methods and technologies
  - Task parallel features were added in OpenMP 3.0
  - Current roadmap suggests that accelerator directives will be added in OpenMP 4.0

Quotes from TBB website FAQ

- Note – Intel not only provides TBB but it also has many members, directors and officers of the OpenMP forum
- Here are some suggestions or tips provided on the TBB website
  - "Everyone should use OpenMP as much as they can. It is easy to use, it is standard, it is supported by all major compilers, and it exploits parallelism well."
  - "OpenMP *[is]* the standard way to do parallelism in C and Fortran."
  - "Use OpenMP if the parallelism is primarily for bounded loops over built-in types, or if it is flat do-loop centric parallelism. … It can be very challenging to match OpenMP performance with TBB for such problems. It is seldom worth the effort to bother – just use OpenMP."
  - "Should I expect TBB to outperform OpenMP and MPI?
    No, TBB may offer a competitive alternative but in general TBB exists to help where OpenMP cannot, and to be far easier to program than MPI."
  - "OpenMP and MPI continue to be good choices in High Performance Computing applications; TBB has been designed to be more conducive to application parallelization on client platforms such as laptops and desktops, going beyond data parallelism …"
- TBB might be a good choice for C++ programmers if their code does not fit the standard data-parallel model
- Note also that TBB and OpenMP can coexist

The Components of OpenMP

- The OpenMP standard defines
  - **Compiler directives** for describing how to parallelise code
    - This is what people normally think about when referring to OpenMP
  - An API of **runtime library routines** to incorporate into your Fortran or C/C++ code for more fine grained control
    - An OpenMP enabled code might not need to use any of these
  - A set of environment variables to determine the runtime behaviour of your application
    - Most people will at least use the environment variable OMP_NUM_THREADS
- An effective OpenMP implementation requires
  - A compiler that can understand the directives and generate a thread-enabled program
    - Most Fortran and C/C++ compilers can understand OpenMP (but check which version of OpenMP they support)
  - An OpenMP runtime library
  - System software that can launch a multi-threaded application on a multi-core system
    - In particular a threaded operating system and tools for mapping processes and threads to multi-core machines

Quick note on GPU accelerator programming

- Coding in OpenMP in order to exploit parallelism on standard multi-core CPUs can deliver advantages – it is worthwhile doing for this reason alone
- OpenMP can also begin a natural progression towards *directive-based* accelerator programming on GPUs
- There are currently directive-based approaches to GPU accelerator programming from PGI and CAPS/HMPP
- A set of directives for accelerator programming are scheduled to be included in the OpenMP 4.0 standard
  - Current plan (as of June 2010) is that a draft standard be available by SC11 (November 2011)
  - Main authors of proposal include PGI and Cray