Distributed and Parallel Technology

Admin Info and Learning Objectives

Hans-Wolfgang Loidl
http://www.macs.hw.ac.uk/~hwloidl

School of Mathematical and Computer Sciences
Heriot-Watt University, Edinburgh

[0] No proprietary software has been used in producing these slides

Motivation

Why tackle the difficult topic of parallel programming?

- Software is no longer sequential.
- Many programs now have to be executed on several computing engines in parallel.
- Parallel machines are getting very diverse (multi-cores, distributed systems, GPGPUs, accelerators etc).
- This course *explores several technologies* that enable the programming and use of such parallel and distributed systems.

Admin Info

Here the basic admin info about the course:

- Lecturers:
  - Hans-Wolfgang Loidl <H.W.Loidl@hw.ac.uk> (EM G48)
  - Sven-Bodo Scholz <S.Scholz@hw.ac.uk> (EM G27)
- Timetable:
  - Mon 10:15 EM 3.06 Lecture
  - Thu 15:15 EM 3.03 Lecture
  - Thu 17:15 EM 2.50 Lab

The main course information page for this course (linked from Vision) is: http://www.macs.hw.ac.uk/˜hwloidl/Courses/F21DP

Admin Info (cont'd)

- Pre-requisites:
  - F29OC: Operating Systems & Concurrency, or equivalent
  - Solid C programming skills (there will be a quick C revision)
- Assessment:
  - Exam: 70%
  - Assessed Coursework: 30% (2 pieces; each as a pair project).
  - The assessed coursework will be handed out in Weeks 4 and 10.
  - CW1 will focus on low-level parallel programming technologies.
  - CW2 will focus on high-level parallel programming technologies.
  - Parallel programs will be run on the department's Beowulf cluster.

Hans-Wolfgang Loidl's Lectures

Concepts

- Parallel architectures
  - SIMD vs MIMD
  - shared vs distributed memory
- Parallel programming models
  - implicit (eg. High-Performance Fortran)
  - semi-implicit (eg. GpH)
  - explicit (eg. C+MPI)

Programming C

- Reading, writing, compiling and running C programs
- Profiling and timing sequential C programs

MPI

Concepts

- Explicit parallelism on distributed memory architecture
- Data transfer and control synchronisation through message passing
  - Point to point communication
  - Collective communication

Programming C with MPI

- Reading, writing, compiling and running simple MPI programs
- Point to point communication in MPI
- Collective communication in MPI
- MPI datatypes

Parallel Performance

Concepts

- Timing/profiling parallel programs
  - What to measure, what to exclude
- Speedup
  - absolute vs relative
  - Amdahl's Law
- Impact of communication to computation ratio on parallel performance

Programming C with MPI

- Timing and profiling MPI programs (with varying # processors)
- Plotting timing and speedup diagrams

Parallel Program Design

Foster's Methodology

- Tuned for specific programming model (tasks and channels)
- 4 steps
  - Partition
  - communication
  - agglomeration
  - mapping

Mattson/Sanders/Massingill: Design Patterns

- Not specific to programming model
- 4 pattern spaces (only two relevant to basic algorithm design)
  - Finding concurrency patterns
  - Algorithm structure patterns

Algorithmic Skeletons

Concepts

- Dual nature of skeletons
  - abstracting common coordination patterns
  - implementing coordination patterns (for specific architectures)
- Skeletons as higher-order functions

Skeleton-based Programming Methodology

- Skeletons identify potential parallelism
- Profiling and cost models decide where to actually parallelise
- Program transformations may rearrange skeletons

Concrete Skeletons

- pipeline
- parallel tasks
- task farm
- divide and conquer

Datacenter Computing

Datacenter Architecture

- Cluster of racks of commodity PCs
- Distributed file system
  - high availability through replication

Programming Model

- MapReduce skeleton
  - MapReduce as higher-order function
  - MapReduce implementation (Google, Apache Hadoop)
    - on top of distributed file system

Vector Processing

Concepts

- SIMD processors architecture
- vector registers

Programming Model

- High level: vectorising compilers
- Low level: C + assembly language
- 4 steps to vectorise C program:
  - Identify vectorisable datatypes
    - Ensure proper alignment in memory
  - Unroll loops
  - Generate assembly code for loop bodies