EEE4120F

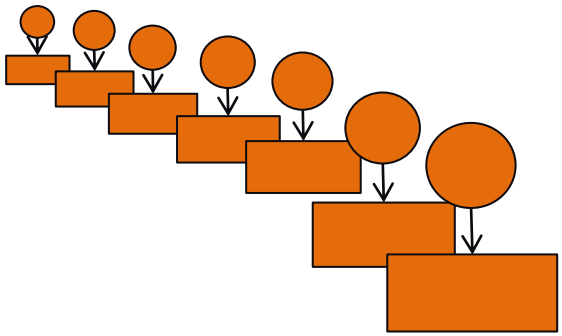
High Performance Embedded Systems

PART 2/3

Lecture 10: Design of Parallel Systems

Presented by

Simon Winberg



Attribution-ShareAlike 4.0 International (CC BY-SA 4.0)

Steps in Designing Parallel Programs

Continuing the path to HPES Applications

Hardcore competent HPES programmers (leading the way to greater feats)

Hey, guys! I forgot my warm jacket, can we go back??

EEE4120F

← Sequential programmers in their comfort zone.

The steps in designing parallel programs

The hardware may be done first… or later.

The main steps:

1. Understand the problem
2. Partitioning (separation into main tasks)
3. Granularity
4. Communications
5. Identify data dependencies
6. Synchronization
7. Load balancing
8. Performance analysis and tuning

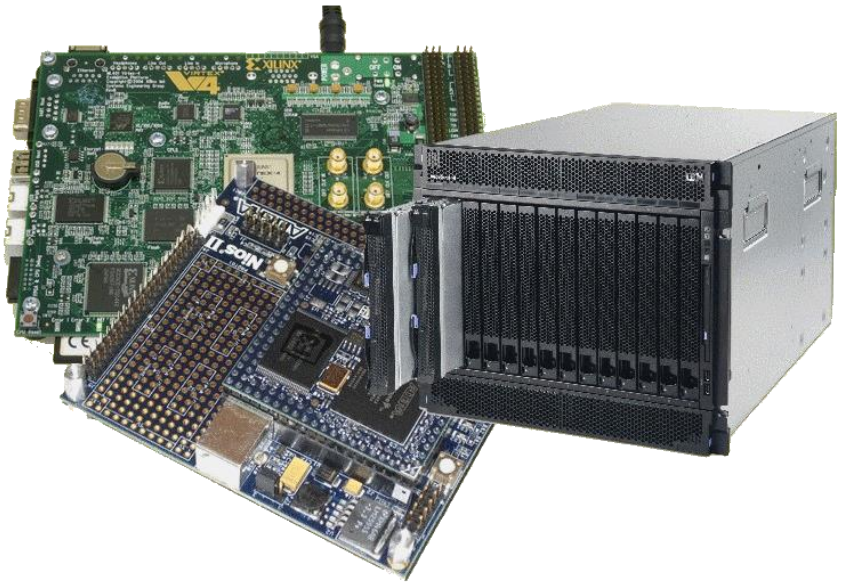
next week

PART 2/3

this lecture

Lecture Overview

- Step 6: Synchronization
- Step 7: Load balancing
- Step 8: Performance analysis and tuning



Steps in designing parallel programs

The hardware may come first or later

The main steps:

1. Understand the problem
2. Partitioning (separation into main tasks)
3. Decomposition & Granularity
4. Communications (later)
5. Identify data dependencies
6. Synchronization
7. Load balancing
8. Performance analysis and tuning

Design of Parallel Programs

Step 6: Synchronization

EEE4120F

Types of Synchronization

- Barrier
- Locking / Semaphores
- Synchronous communication operations

Barrier

- *All* tasks are involved
- Each task does work until it reaches the barrier, and then blocks.
- When the last task reaches the barrier all the tasks are synchronized.
- What happens next?…
  - Section of work is done *or*
  - Tasks are automatically released to continue their work… programmer usually decides.

Locking / Semaphores

- May concern any number of tasks
- Usually used to serialize / protect access to global data or critical section of code.
- Only one task at a time may have the lock / semaphore.
- Tasks can attempt to get the lock need to wait for the task that has the lock to release it, granted on a FCFS basis.
- Usually blocking, could be non-blocking (i.e., do other work until lock is available)

Synchronous communication operations

- Concerns only tasks executing a communication operation (or comms op)
- When a task performs a comms op, some form of coordination is required with other task(s) involved with this communication.
  - For example, before a task can do a send operation, it needs to first receive a clear to send (CTS) signal from the task it wants to send to.

Step 7: Load Balancing

EEE4120F

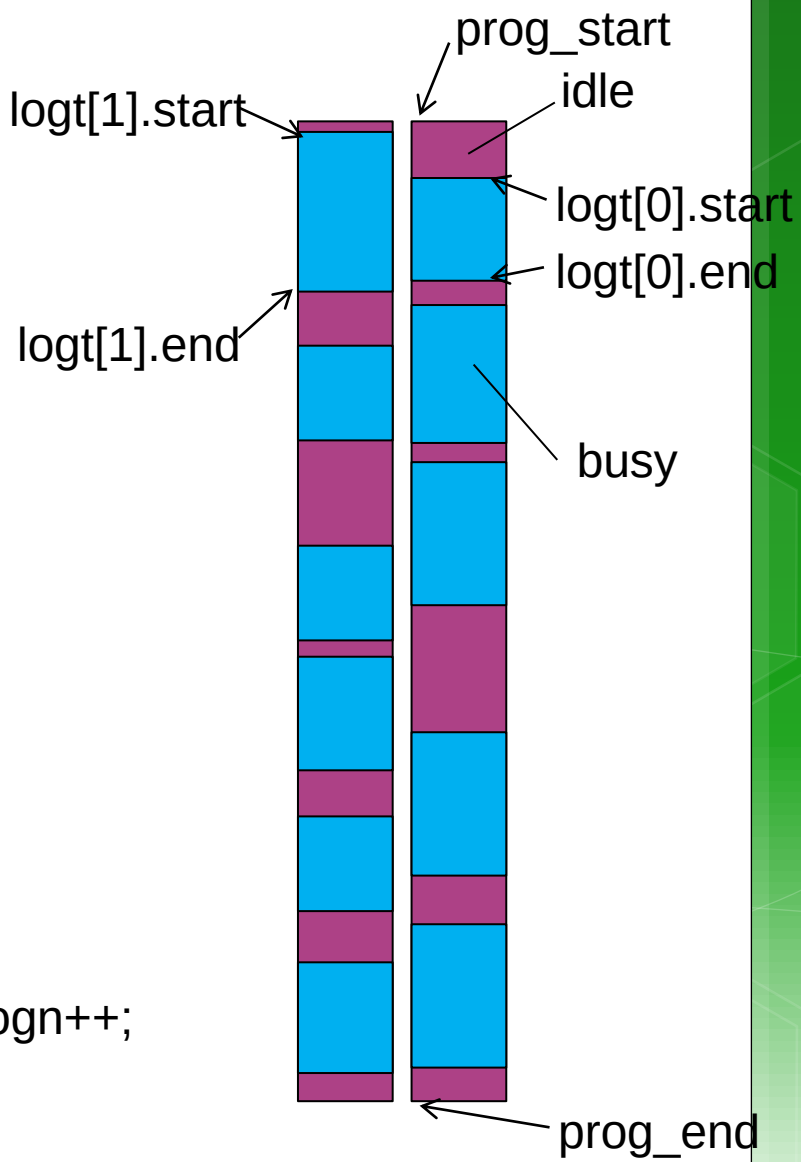
Load Balancing

- Distributing work among tasks so that all tasks are kept busy most of the time
- Mainly a minimization of idle time
- Relevant to parallel programs for optimizing performance.
  - e.g., if every tasks encounters a barrier synchronization point, the slowest task will end up limiting the overall performance.
- Relevant to costing, e.g. deciding number of processors needed (can also be used in substantiating the need for a more, rather than less, expensive platform)

Load Balancing – simple trial technique

```
…
typedef struct { unsigned start, end; } LogEntry;
LogEntry logt[1024];
int logn = 0;
unsigned prog_start, prog_end;

void* task (void* arg)
{
  unsigned int end;
  unsigned int start = clock();
  // do work
  // calculate time it took
  end = clock();
  // update time log
  pthread_mutex_lock(&mutex);
    logt[logn].start = start; logt[logn].end = end; logn++;
  pthread_mutex_unlock(&mutex);
}
```



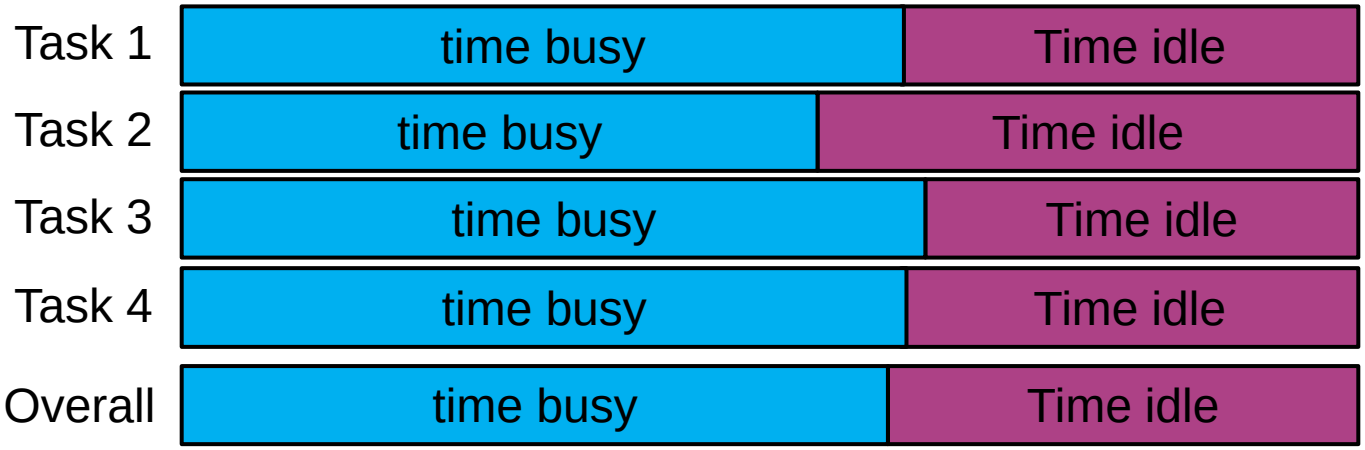
… in main() get prog_start time, create mutex & threads, wait join, get prog_end time.

Load Balancing – simple trial technique (cont)

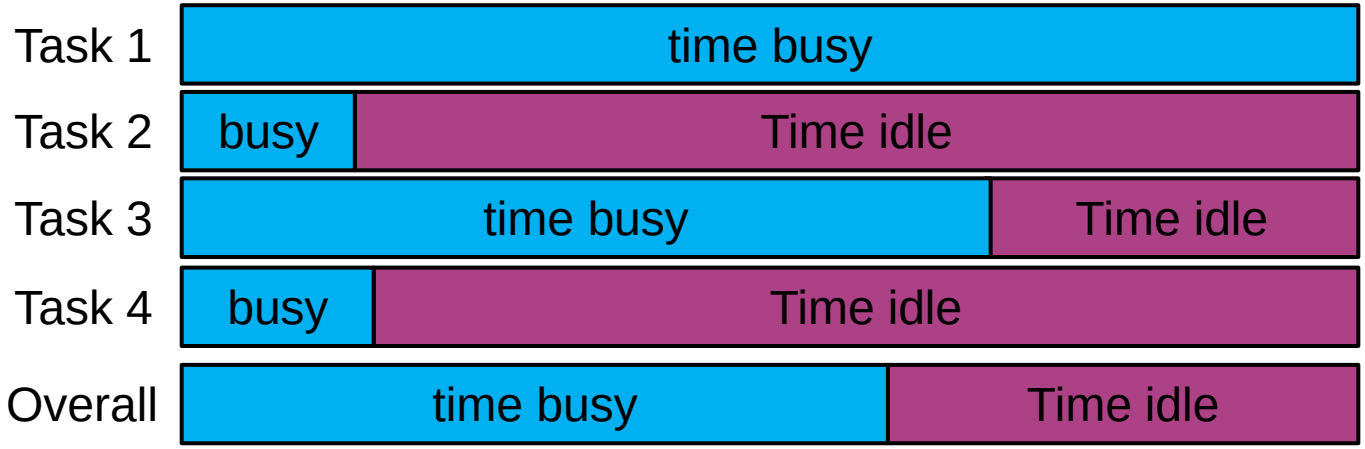
```
...
double calc_stat_stddev ()
{ // calculate standard deviation of busy times
  int i;
  double sum = 0;
  for(i = 0; i < logn; i++) sum += logt[i].end - logt[i].start;
  double mean = sum / logn;
  double sq_diff_sum = 0;
  for(i = 0; i < logn; ++i) {
     double diff = logt[i].end - logt[i].start - mean;
     sq_diff_sum += diff * diff;
     }
   double variance = (double)sq_diff_sum / logn;
   return sqrt(variance);
}
…
```

Load Balancing

Well balanced – low standard deviation in busy time*



Poorly balanced – high standard deviation in busy time



* You could obviously use idle time – but that's probably more difficult if using code in prev. slide

Achieving load balance

- Two general techniques…

1. Equally partition work as a preprocess
   - Workload can be determined accurately before starting the operation
2. Dynamic work assignment
   - For operations where the workload is unknown, or cannot be effectively calculated, before starting the operation

Achieving balance – Work assignment as pre-process

- Examples include
  - Array or matrix operations for which each task performs similar work. Can evenly distribute the data set among tasks / available processors.
  - Loop iterations where the work in each iteration is similar, and can be evenly distributed.

May be less easy to do if using a heterogeneous mix of machines that have varying performance characteristics. In such cases, a comprehensive analysis tool may be needed to determine load imbalances and revise work disctribution.

Achieving balance – Work assigned dynamically

- Examples include
  - Sparse arrays/matrices (i.e., if matrix simply divided equally, some tasks may have lots to do but others little to do)
  - N-body simulations (particles owned by some tasks may involve more work than others)
  - Adaptive grid refinement (some tasks may need to refine their grips)

General principle: tasks that are busier send a request for assistance, whereas tasks that complete quickly are available to help the busier tasks.

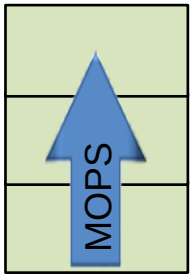
Vertical vs. horizontal load balancing

Recommended video clip to view

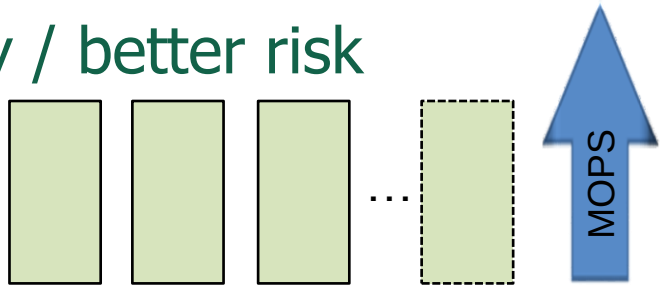
Load Balancing with Cloud Computing.mp4

Vertical vs. horizontal load balancing

- Vertically:
  - More power, but centralized (one app. may run faster / more response / less latency for RT)
  - Adding CPUs + memory to one system
  - More dependent on one machine / group of closely coupled machines



- Horizontally:
  - More power, but wider distribution (more apps able to run at one time)
  - Simpler scalability
  - More distribution / redundancy / better risk management



Step 8: Performance analysis

EEE4120F

Performance analysis

- Simple approaches involve using timers (i.e., as shown in previous code snippet)
  - Usually quite straightforward for a serial situation or global memory
- Monitoring and analyzing parallel programs can get a whole lot more challenging than for the serial case
  - May add comms overhead
  - Needing synchronization to avoid corrupting results, … could be a nontrivial algorithm itself

MPI Timing functions

NB: MPI optional in 2023

- MPI_Wtime
  - Returns the *elapsed wall clock time* in seconds (a double) on the calling processor
- MPI_Wtick
  - Returns the *timer resolution* in seconds (a double) of MPI_Wtime.

Note: MPI is the topic of a later lecture.

StdC: gettimeofday

- **gettimeofday**
  - **Very portable, part of the StdC library**
  - **Should be available on any Linux system**
  - **Returns time in seconds and microseconds since midnight January 1 1970**
  - **Uses struct timeval comprising**
    - **tv_sec : number of seconds**
    - **tv_usec : number of microseconds***
  - **Converting to microseconds will use huge numbers, rather work on differences**

**** Word of caution: some implementations always return 0 for the usec field!!***
On Cygwin, the resolution is only in milliseconds, so tv_usec in multiples of 1000.
Not provided in DevC++.

gettimeofday example

see **timing.c** code file
Provided previously

```
#include <stdio.h>
#include <sys/time.h>
#include <time.h>
struct timeval start_time, end_time; // variables to hold start and end time

int main()  {
   int tot_usecs;
   int i,j,sum=0;
   gettimeofday(&start_time, (struct timezone*)0); // starting timestamp

   /*   do some work */
   for (i=0; i<10000; i++)
      for (j=0; j<i; j++) sum += i*j;

   gettimeofday(&end_time, (struct timezone*)0);  // ending timestamp

   tot_usecs = (end_time.tv_sec-start_time.tv_sec) * 1000000 +
               (end_time.tv_usec-start_time.tv_usec);
   printf("Total time: %d usec.\n", tot_usecs);
}
```

Benchmarking

- Process of measuring performance of DS
- A program that Quantitatively evaluates computer Hardware and software resources
- Benchmark suites – sets of benchmarks defined to get better system picture
- Suitable benchmark for a system leads to an effective system

Benchmarking (cont…)

(Recap mostly, see earlier lecture)

Supplementary reading

- What can be benchmarked? (In DSP Compiler, Processor, Platform)
- Compiler – Converts High Level Language to Assembly language thus we benchmark compiler efficiency
- The Processor – code should be an Assembly (No more high level code)
- Platform – Written in High Level language(Processor & Compiler)

Benchmarking: what to measure

Supplementary reading

- In DSP/HPEC systems we benchmark Cycle count, Data and Program memory usage, Execution time and Power consumption (This point mentioned already, next two are new…)
- What can we benchmark in Databases/Web servers (i.e Oracle, MySQL, SQL Server)?
- SPEC defined several benchmarks in the digital world - SPECweb99, SPECmail, etc.

Benchmark: Approaches

- Metrics (MIPS, MOPS, MFLOPS, etc.) – some of these not really meaningful in RISC architectures (why?)
- Complete DSP Application
  - consumes time and effort as well as memory because it measure the whole system
- DSP Algorithm Kernels
  - extracted from real DSP programs and consist of small loops which perform number crunching, bit processing, etc.

Benchmarking: Types

Supplementary reading

- EEMBC (pron. embassy) – For embedded System and written in C
- BDTIMARK – a DSP benchmark suite
- Check this paper for a List of FPGA benchmarking types: Raphael Njuguna: A Survey of FPGA Benchmarks Available at: http://www.cse.wustl.edu/~jain/cse567-08/ftp/fpga.pdf) -- might be in exam
- More Reading: Performance evaluation and Benchmarking, Lizy K. John and Lieven Eeckhout, Taylor & Francis Group

Part 2 Done

6. Synchronization
7. Load balancing
8. Performance analysis and tuning

Over to you to work on OpenCL prac or have an early start on Prac3

Questions?

Further Reading / Refs

Suggestions for further reading, slides partially based / general principles elaborated in:

- A Survey of FPGA Benchmarks Available at: http://www.cse.wustl.edu/~jain/cse567-08/ftp/fpga.pdf
- Data dependencies suggested reading: http://sc.tamu.edu/help/power/powerlearn/html/ScalarOptnw/tsld036.htm
- Load Balancing with Cloud Computing. https://www.youtube.com/watch?v=_xH5POea0Jc
- FPGA benchmarking types: Raphael Njuguna: A Survey of FPGA Benchmarks Available at: http://www.cse.wustl.edu/~jain/cse567-08/ftp/fpga.pdf
- “Performance evaluation and Benchmarking”, Lizy K. John and Lieven Eeckhout, Taylor & Francis Group

Disclaimers and copyright/licensing details

I have tried to follow the correct practices concerning copyright and licensing of material, particularly image sources that have been used in this presentation. I have put much effort into trying to make this material open access so that it can be of benefit to others in their teaching and learning practice. Any mistakes or omissions with regards to these issues I will correct when notified. To the best of my understanding the material in these slides can be shared according to the Creative Commons “Attribution-ShareAlike 4.0 International (CC BY-SA 4.0)” license, and that is why I selected that license to apply to this presentation (it’s not because I particulate want my slides referenced but more to acknowledge the sources and generosity of others who have provided free material such as the images I have used).

Image sources:

Man running up stairs - Wikipedia (open commons)
Chocolate Chip Biscuit - Wikipedia (open commons)
Scales, Checked Flag - Open Clipart (http://openclipart.org)
Calendar Planning Image, Note pad – Pixabay (Public Domain CC0)
Yoga lady - adapted from Open Clipart

Disclaimers and copyright/licensing details

I have tried to follow the correct practices concerning copyright and licensing of material, particularly image sources that have been used in this presentation. I have put much effort into trying to make this material open access so that it can be of benefit to others in their teaching and learning practice. Any mistakes or omissions with regards to these issues I will correct when notified. To the best of my understanding the material in these slides can be shared according to the Creative Commons “Attribution-ShareAlike 4.0 International (CC BY-SA 4.0)” license, and that is why I selected that license to apply to this presentation (it’s not because I particulate want my slides referenced but more to acknowledge the sources and generosity of others who have provided free material such as the images I have used).

Image sources:

Clipart sources – public domain CC0 (http://pixabay.com/)
PxFuel – CC0 (https://www.pxfuel.com/)
Pixabay
commons.wikimedia.org
Images from flickr