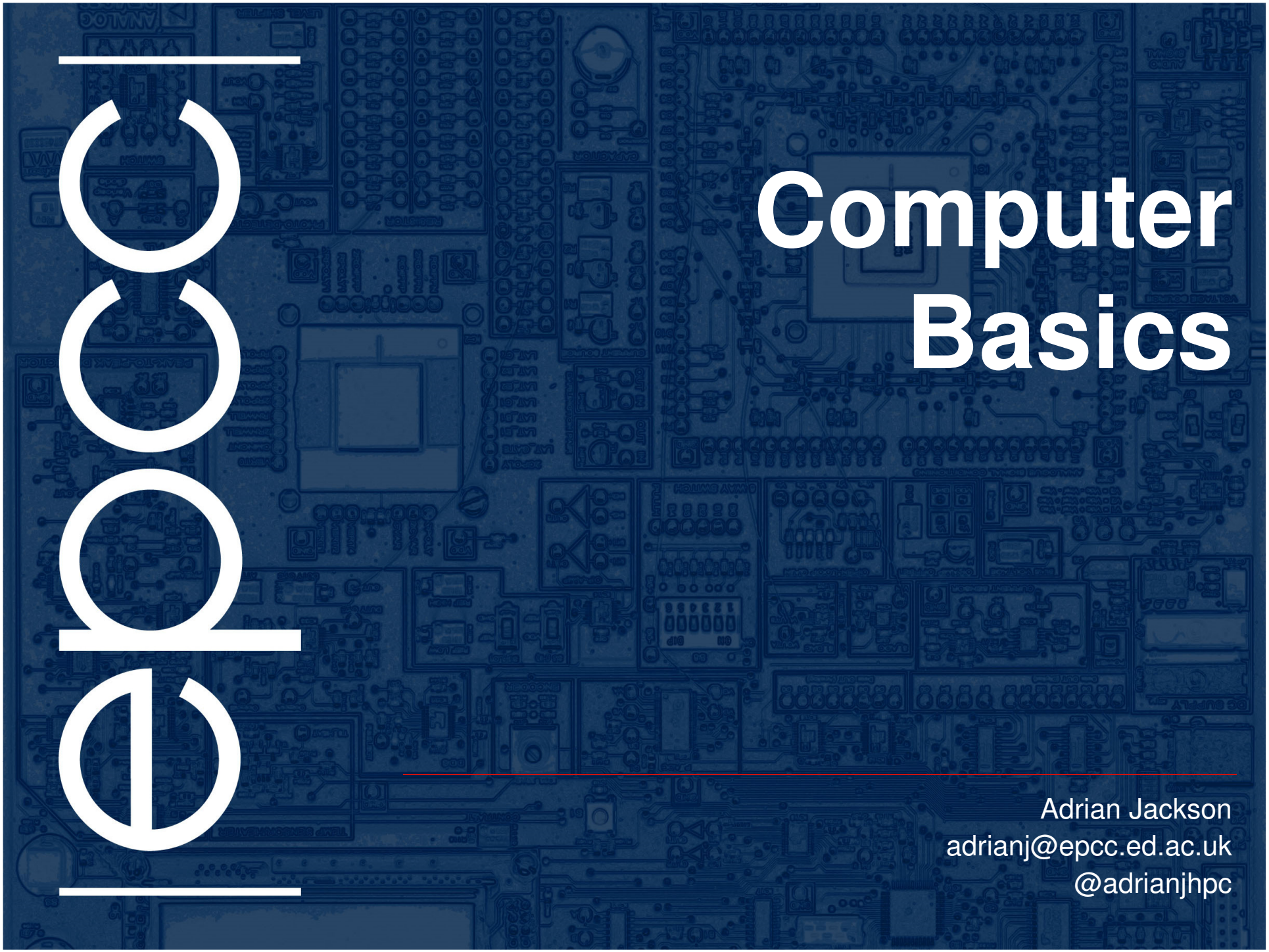


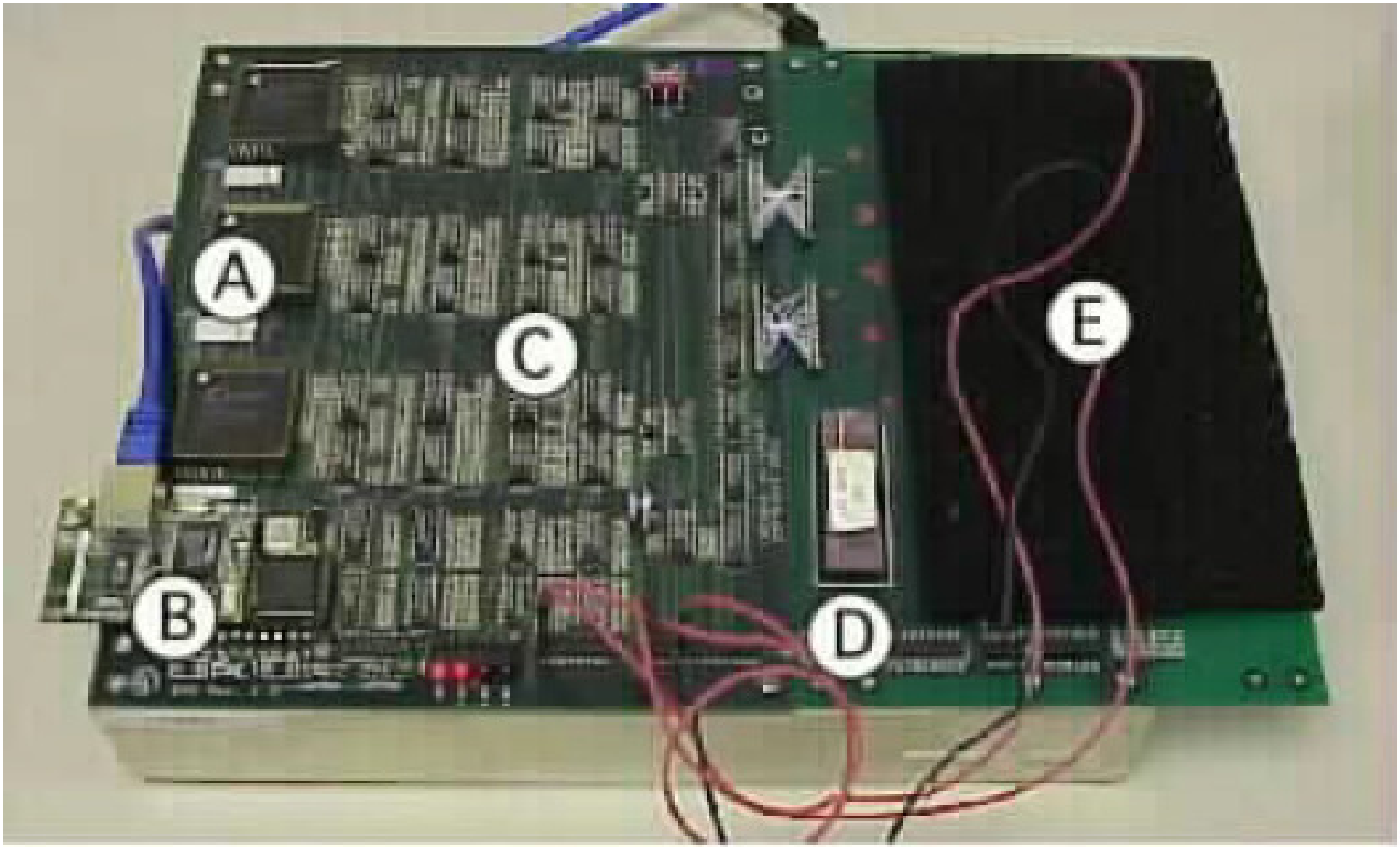
Computer Basics

Adrian Jackson
adrianj@epcc.ed.ac.uk
@adrianjhpc

What is a computer?

- Device that manipulates data according to a set of instructions
  - Simple mathematical operations
  - Many operations per second
  - Can be programmed (i.e. instructions sets than can be saved and used when needed)
  - Embedded most common, Personal Computer what most think of as a computer
- Digital
  - Information stored in discrete quantities
  - 0 and 1 represented by presence or absence of electricity
  - Easy to create electronic circuits based on this

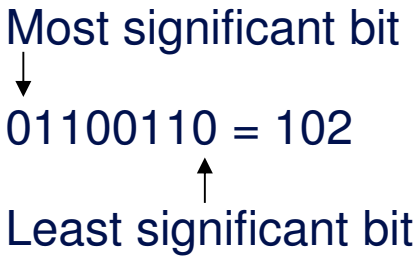
Analog Computers



Analog Computers

Basic concepts

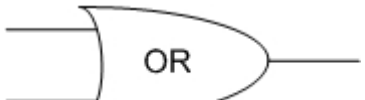
- Binary data
  - Based on 0 and 1 digits
  - Useful as easy to build components that have 2 states (i.e. transistor conducting or non-conducting)
  - Base 2 with each digit 1 bit
  - Position of digit determines value

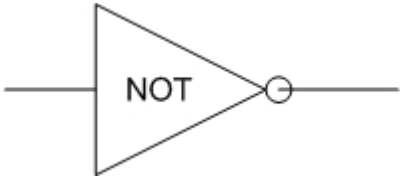


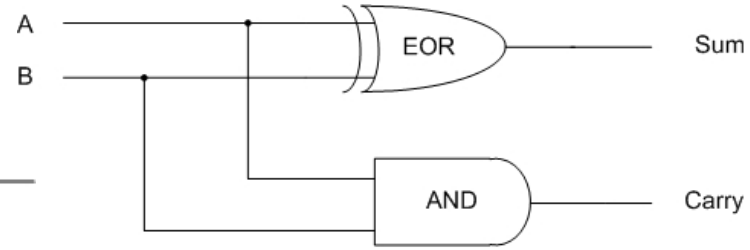
- Further refinement for quicker processing or large datasets or safety
  - Sign bit
  - Twos complement number
  - Parity checking
- Groups of bits have different names:
  - Nibble: 4 bits
  - Byte: 8 bits
  - Word: Dependent on machine architecture

Basic concepts cont.

- Boolean logic used to combine binary data
- Some fundamental boolean operations
  - NOT
  - AND
  - OR
  - EOR (Exclusive OR)
  - NAND
  - NOR







A	B	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

- Boolean operations built as logic gates in hardware
  - Can be connected together to build more complex functionality i.e. half adder
  - Also necessary for storage (i.e. memory built from flip-flops)

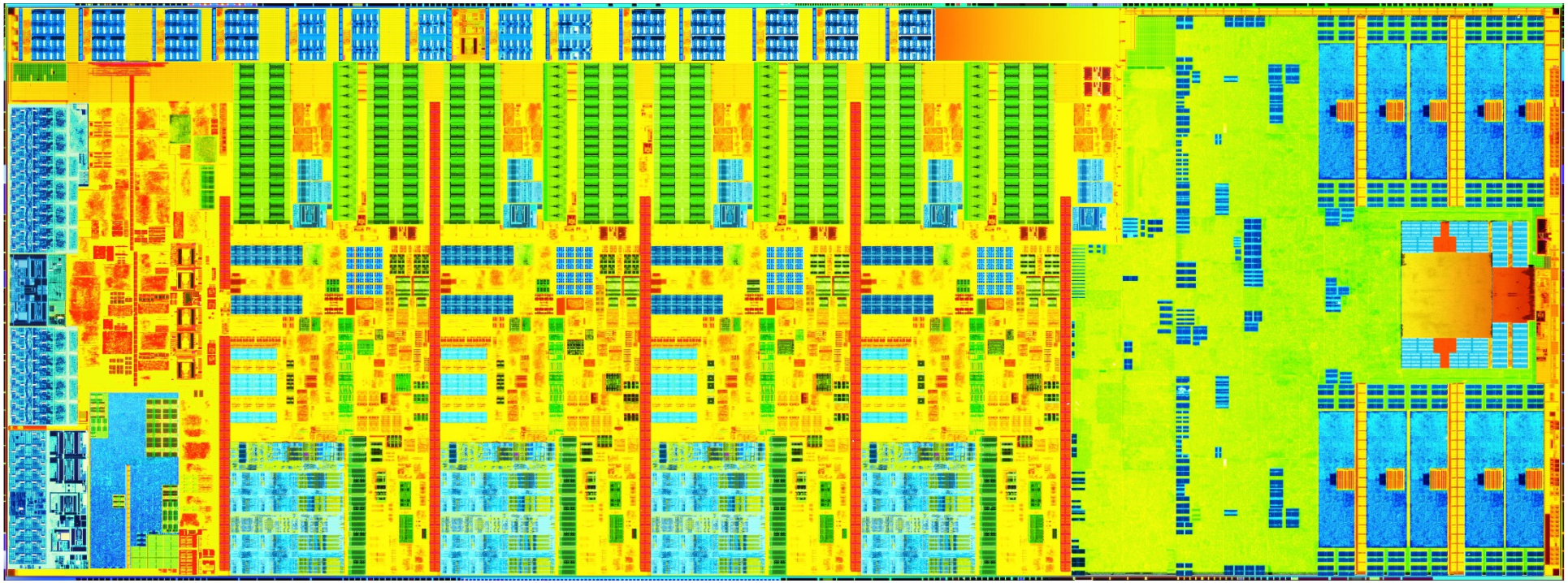
Basic components

- Main groups:
  - CPU
  - Memory
  - Input and output devices (I/O)
- Interconnected by buses
  - Groups of wires (examples IDE, PCI, Serial, PCI-X)
- CPU
  - Based on simple logic, grown very complex over last 60 years
  - Can actually contain many processing elements now
  - Contains a number of distinct parts:
    - Control Unit
    - Clock
    - Arithmetic/Logic Unit (ALU) (including pipelines)
    - Registers
    - Cache
    - Floating Point Unit (FPU)

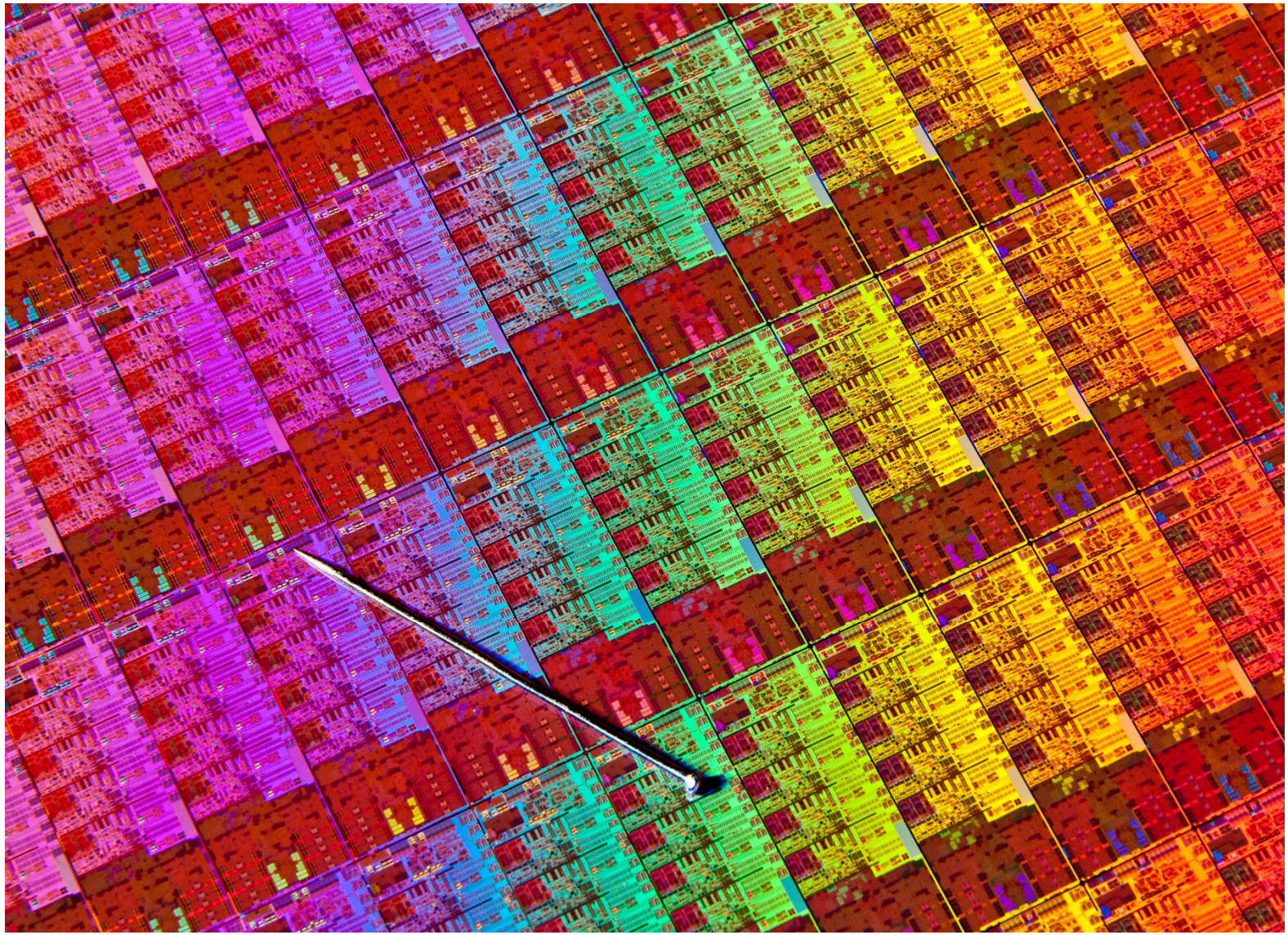
CPU cont.

- Control Unit
  - Fetches, decodes, and executes instructions
  - Controls pipelines, schedules execution, manages outputs
- Cache
  - Fast storage, on chip, allow re-use and pre-fetch of data
  - Both instruction and data caches (TLB as well)
  - Complex cache hierarchies
- Registers
  - Small amount of fast, close, memory
  - Lots of different kinds, processor dependent
- ALU, FPU, and Instruction Pipelines
  - Integer, floating point, and program flow arithmetic
  - Often pipelined to improve performance
  - Example RISC pipeline:
    1. Instruction fetch
    2. Instruction decode and register fetch
    3. Execute
    4. Memory access
    5. Register write back

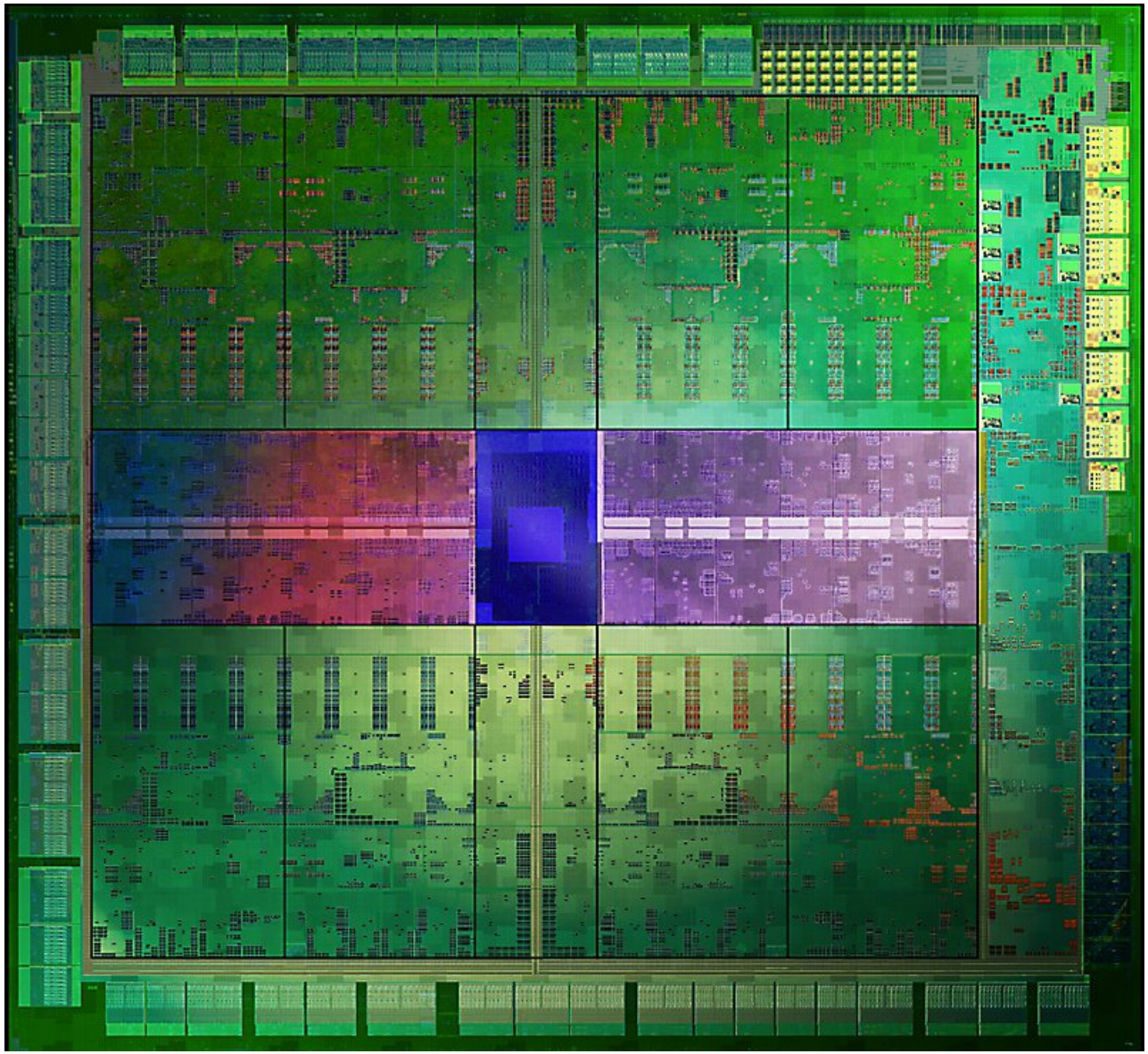
CPU Example – Intel Quad Core Haswell Die 22nm



Haswell die scale

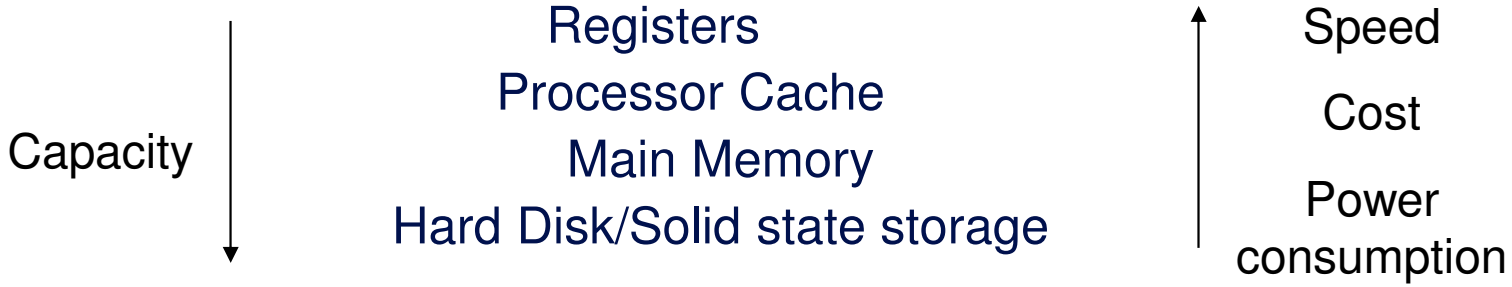


NVIDIA Tesla K20 (GK110) GPU



Memory and I/O Devices

- Modern computers have memory hierarchy
  - O/S allows virtual memory



- Massive difference in performance
  - Registers operate at clock speed
  - Main memory fetch can be 10-100s times slower
  - Disk fetch 1000s times slower
  - Processor need data to do work
- Memory latency and complex pipelines can reduce processor performance
  - 10-25% of nominal peak

Disks, Storage, and Summary

- Large storage necessary for programs and data
  - Hard disk drives connected by bus (IDE, SATA, SCSI, etc…)
  - Solid state storage
  - Network attached storage (i.e. nfs)
- Different technologies have different characteristics
  - NAS accessible from many machines but slow
  - Hard disk can be fast but static
  - Solid state removable but expensive
- Computers may use a number of different storage technologies/solutions
- Computers complex set of systems
  - Large number of optimisations to processors and memory
  - Wide range of peripherals
- Aim to keep processor as busy as possible

Software

- Anything that runs on a computer or electronic device
  - Instructions for the CPU and other devices
- Different levels:

Machine Code	System Software	Platform Software
Assembler Code	Programming Software	Application Software
Programming Languages	Applications	User-written Software

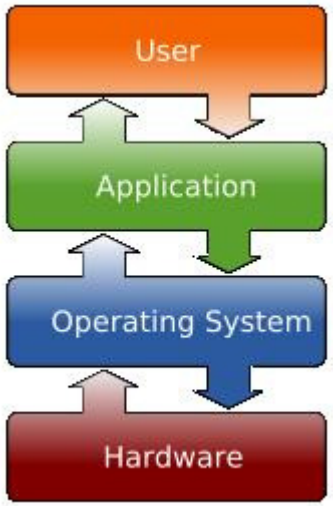
- May use libraries or specific hardware
- Many different characteristics and requirements
  - Portability
  - Usability
  - Performance
  - Design
  - Etc…

Operating System (O/S)

- Underlying program (or collection of programs) on computer
  - Controls and runs the computer
- Basic job is to provide platform for programs to run
  - Manage system resources
  - Manage running programs
  - Provide user interface
  - Abstract hardware details from programmer and user
- Many programs run concurrently
  - Some O/S allow multiple users
  - This includes O/S and user programs
  - Often called processes and threads
- Memory management and Program Execution
  - Program memory
  - Virtual memory

O/S cont.

- Deals with hardware and software communications
  - Interrupts from hardware and programs
  - Allows multi-tasking, time sharing …
- Different ways of categorising Operating Systems
  - Multi-user vs. Single user
  - Multitasking
  - Networked vs. standalone
  - Real-time Systems
  - Embedded
- 2 basic kinds for our purpose:
  - Windows based (GUI)
  - Command line based (CLI)
- Distinctions not rigid, but is important
  - Command line based have Windows managers
  - Windows based will have some command line functionality
  - GUI based less useful for HPC



O/S Environment

- Device drivers and hardware specifics dealt with by O/S
  - Allows many different hardware items to use same hardware connections
  - Bridge between O/S and vendor hardware
- HPC computers run linux or unix
  - Command line based
  - Multi-user platforms
  - Often different compute elements
- O/S protects system from user, allows user to run programs, manages system resources
  - Allocates time, disk spaces, etc.. to users
  - Manages swapping programs on processor
- Run as normal user
  - Administrators are generally super users
  - Protects O/S and other users from each other
- Distinguished by username and password
  - Assigned user id and group id

O/S Environment cont.

- Provided “home” directory
  - /home/adrianj
  - Can do anything within that directory
  - May be quotas/limits
- Generally can access own directory and other directories of similar group
  - O/S often uses file permissions

```
drwxr-xr-x   4 adrianj  epcc          5 Feb 18  2004 Federation
drwxr-xr-x   4 adrianj  epcc          4 Mar 11  2004 presentations
-rwxr-xr-x   1 adrianj  epcc       1730 Mar 11  2004 presentations.html
-rwxr-xr-x   1 adrianj  epcc       1446 Jul 13  2004 index.html
drwxr-xr-x   2 adrianj  epcc          3 Oct  7  2004 Benchmarking
drwxr-xr-x   3 adrianj  epcc         10 Apr 24 12:26 ePortal
-rwxr-xr-x   1 adrianj  epcc       1239 Apr 24 12:26 projects.html
-rwxr-xr-x   1 adrianj  epcc       1188 Apr 24 12:30 bio.html
drwxr-xr-x   2 adrianj  epcc          3 Sep  3 17:52 photos
```

O/S and the user

- Can run programs from home space
  - Can require ./ prior to executable name to execute
  - Programs run as processes managed by O/S
- Each user can customise their environment to select programs
- Often access is via remote login
  - Secure Shell (SSH)
  - File transfer over SSH (SCP/SFTP) or network mount point
  - Graphics display becomes an issue
- X windows systems often used
  - Allows windows to be displayed remotely
  - Graphics over SSH
  - Has performance issues

Program

- Program is a text or group of text documents
  - When compiled an executable is produced
- A process is an instances of a running executable:
  - Set of instructions
  - Memory space
  - Context – registers, program counter, memory addresses, …
  - O/S descriptors – i.e. file handles, data sources/sinks, ….
- Memory space has different parts
  - Heap: Dynamic allocation area
  - Stack: Return addresses, local data storage, parameter passing
- Heap – memory available through-out program
- Stack – automatic memory, often only available within calling scope

Programming Language

- Program is a method for creating stored instructions to be executed at a later time
  - All operations on a computer can be classed as programs
- Programming in instructions very difficult, very machine orientated
  - Assembler programming very machine specific
  - Large number of lines for simple operations
  - Unreadable and easy to make mistakes
  - May get better performance
- Programming Languages are application or problem oriented approach
  - Portable, machine independent (as long as compiler available)
  - More concise, good control and data structures available
  - Make programming computers easier and more robust
  - Provides optimisation opportunities (let compiler do the work)

Programming Language cont.

- General aims:
  - Application/problem orientated
  - Machine independent
  - Clear structure: Able to express problem
  - Simplicity: Good set of basic operations
  - Efficiency: Good machine code
  - Readability: High cost in maintaining/debugging/optimising programs
- Components:
  - Language definition or syntax
  - Compiler or interpreter

Assembler vs Programming language

C code:

```
for(i=0; i<100; i++){
  a += i;
}
```

Assembler Code:

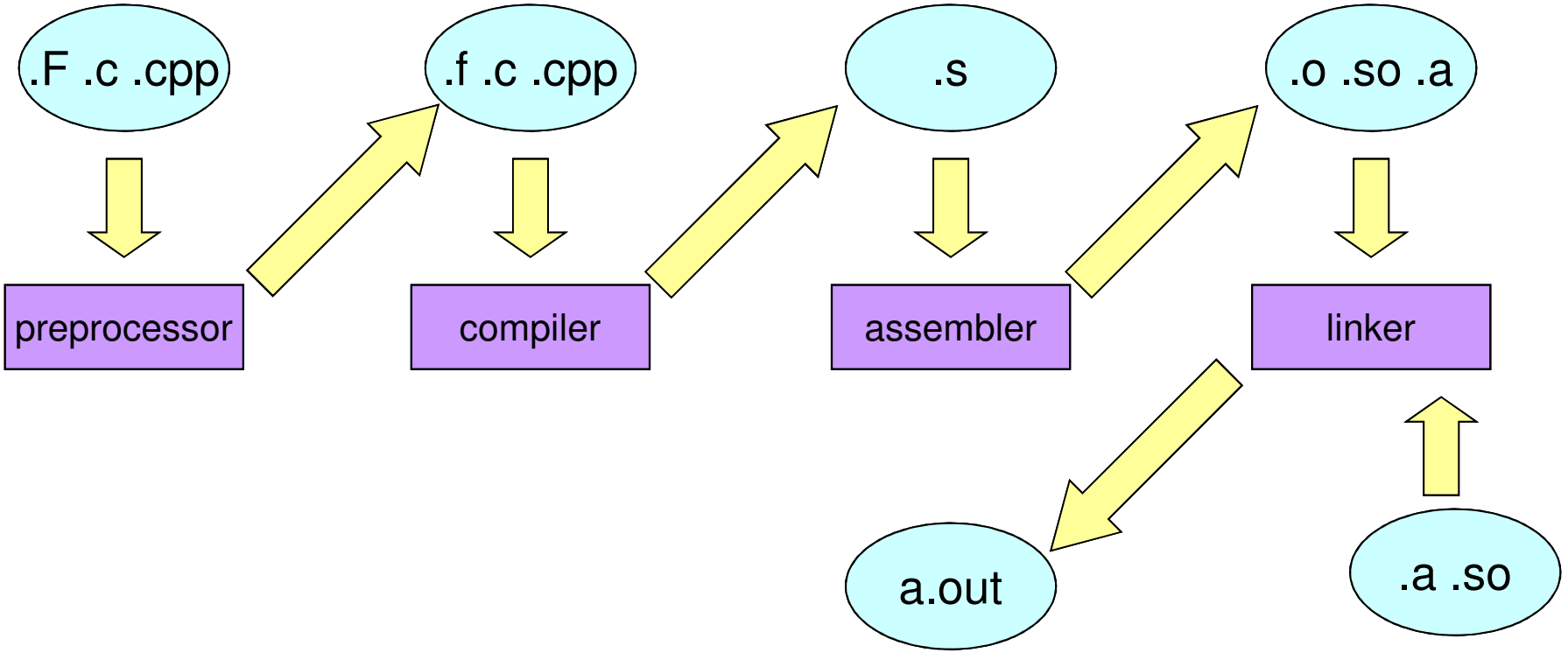
```
mov     %g0,%i5
add     %i4,%i5,%l2
mov     %l2,%i4
add     %i5,1,%i5
cmp     %i5,100
bl      .L96
nop
```

Code compilers

- Fundamental tools of the trade:
  - Code preprocessors (e.g. **`cpp`**, **`fpp`**, **`gpp`**)
  - Code compilers (e.g. **`f77`**, **`f90`**, **`cc`**, **`CC`**, **`c++`**, **`gcc`**, **`javac`**)
  - Code linkers (e.g. **`ld`**, **`ild`**)
- Knowing how these things work is valuable
  - Understand optimisation
  - Understand how to use correctly in, e.g., makefiles
  - Understand how your program will work on strange architectures
- Joe BankDBProgrammer may not need to know
- Sally HPCWizard does

What do they do?

- Translate high-level language into machine runnable code



Preprocessors

- Preprocess source code before compilation
  - Use to define constants and macros (e.g. **`#define TRUE 1`**)
  - Also to include other code files (e.g. **`#include <stdio.h>`**)
  - Also for conditional compilation (e.g. **`#ifdef PAR_MPI`**)
- Most common is C preprocessor **`cpp`**
  - Run implicitly by C compilers
  - Also run implicitly by most Fortran compilers for **`.F`** / **`.F90`**
    - Though some use slightly different versions (e.g. **`fpp`**)
  - Can be used explicitly for any language
    - Makefile is good place to define preprocessing rules like this:

```
.texs.tex:
        cpp $< > ${<:.texs=.tex}
```

Preprocessors

```
void s_fact(void)
{
#ifdef MFACT_TIMES
  double t0, t1;
  t0 = mfact_clock();
#endif

.... body of function in here....

#ifdef MFACT_TIMES
  t1 = mfact_clock();
  fprintf(logfile_p, "s_fact time ",t1-t0);
#endif
}
```

Preprocessors

- Preprocessors are a powerful and useful tool
- **But:** avoid overuse...
- Don't use macros for the sake of it
  - Macros should enhance readability as well as (possibly) performance
- Don't use macros that extend over many lines
  - Can confuse debuggers as to correct line numbering
- Don't use too many `#ifdef`s in one function
  - If you're conditionally switching a lot of code, you probably should write it as a completely separate function

Preprocessors

- Be careful with magic numbers
  - No typechecking is done for **`#define`**'d constants

```
#define IN_BUFFER_SIZE 20
#define OUT_BUFFER_SIZE 30
:
char in_buffer[IN_BUFFER_SIZE];
```

- Can use other types for this
  - Helps debuggers

```
enum {
  IN_BUFFER_SIZE  = 20,
  OUT_BUFFER_SIZE = 30
};
```

- Can get compiler to show source code after preprocessing performed

```
gcc -E
```

Compilers

- Take (preprocessed) source code and generate assembly language
- Usually work in two distinct stages
  - Syntactic and semantic analysis (front end)
    - Checks your code for “correctness”
    - Generates intermediate language representation
  - Assembly code generation (back end)
    - Takes intermediate language and generates assembler
- Can incorporate optimisation in either or both stages

Assemblers

- Turn assembly language into object code
- Very architecture/machine specific
- Can include optimisation
- Assembly is usually a hidden stage in compilation
  - Unless you have to write explicit assembler!
- Sometimes necessary to write assembler, or be able to examine the assembler produced
  - Optimise particular sections
  - Check the compiler is doing the correct thing
  - `gcc -S` will produce assembler

Linkers

- Put all the pieces together to make a program
- Unite all the object files and join them to libraries
  - Library locations defined by flags (e.g. **`-L`**) or by “library path” environment variable (e.g. **`LD_LIBRARY_PATH`**)
- Add operating system “hooks” for calling dynamic libraries
- Bind all symbolic references to memory addresses
  - Identify the “main” routine as the initial entry point
- Some linkers work incrementally (e.g. **`ild`**)

Compiler systems

- All the preceding phases are generally presented as one "compilation process"
  - e.g. **`f90 mycode.F90`** actually does all this for you (Solaris):
    - **`fpp mycode.F90 → mycode.for.f90`**
      `f90 -s mycode.for.f90 → mycode.s`
      `as mycode.s → mycode.o`
      `ld mycode.o <libraries> → a.out`
- Specify compile options for any of these phases
- Understanding the options available for your compiler is *essential*, especially when optimising
  - Use the documentation
  - Can affect both performance and results!

Compiler systems cont.

- Choice of compiler can affect performance
  - GNU compilers (i.e. gcc, g++, gfortran) usually always available
  - Machine specific or commercial compilers often give better performance
- Choice of compiler "flags" essential to good performance
  - Development and debug should be compiled differently to production runs
  - Optimisation level can improve performance (i.e. `-O5`)
  - Some optimisations can change numerical accuracy
  - Flags set architecture being used, HPC programming often uses different machines to compile and run the code
  - Debug information and performance information often needs compiler flags (i.e. `-g -pg`)
- Read the man pages or documentation of the compiler you're using
- Default compilers (i.e. gcc) can allow "bad" code by default
  - Problem for portability, look at compile options to restrict such behaviour
- Single file program the following is usually fine

```
gcc -O3 -o name program.c
```

Libraries

- A library is just a collection of object files
  - A useful (or essential!) collection of functions, subroutines or classes
- Libraries are A Good Thing
  - They promote clean, well-behaved interfaces between logically separate chunks of code
  - The promote reuse and discourage the reinvention of the wheel
  - Save a lot of time and effort
  - May have dealt with portability problems
  - Special-purpose libraries are often optimised for the current architecture
    - Never write your own FFT...
- Libraries have two broad types:
- Static libraries (`.a` or `.lib`)
  - Library code copied into your program
  - Large executable file but can give marginally faster code
  - Wasteful: each user's program has its own copy of the library
- Dynamic libraries (`.so` or `.dll`)
  - Library linked dynamically, i.e. at runtime
  - Smaller executable file
  - Single library copy can be used by multiple running programs
  - Pick up latest (improved!) versions when you run
    - Though can lead to non-bit reproducibility
  - Changes of library locations or versions → cryptic runtime failures

Batch Systems

- Batch systems equate to Job Schedulers
- Most HPC systems have 2 parts
  - Front-end: Login nodes for users, compilation resource, filesystem, small numbers of processors
  - Back-end: Production system, large numbers of processors
- Important for any HPC system
  - Software for managing and launching executables on back-end hardware
  - Used to separate jobs from hardware
  - HPC jobs require dedicated machine access
  - Proprietary software, a number of different systems around
    - Sun Grid Engine
    - Loadleveler
    - PBS
    - LSF

Batch Systems cont...

- Used to protect hardware, secure systems, optimise machine usage, enforce usage policies
  - Very configurable by administrator/operator
  - Some choice for user
- Often back-end systems do not allow direct logon
  - No public ip addresses, not running same level of O/S
  - Even if they do, need a system for running jobs across large numbers of processors/machines/frames/etc…
- Basic features include:
  - Automatic submission of jobs
  - Interfaces to monitor execution
  - Priorities and/or queues to control the execution order of unrelated jobs
  - Interfaces which helps to define workflows and/or job dependencies
  - Scheduling optimisation algorithms
  - Resource monitoring/accounting

Batch System Concepts

- Queues
  - Portions of machine and time constraints
  - Generally small numbers of defined queues
- Generally specify:
  - Executable name
  - Account name
  - Maximum run time
  - Number of CPUs
  - Output file names/directories

HPCx (phase3) batch system LPAR allocation status at: 2008-10-20 14:24:30

	11	12	13	14	15	16	17	18
f401	(cm)	(inter)						
f402	(serial)	(inter)						
f403	.....	uclmbw	dlrojo	uclmbw	uclmbw	uclmbw	uclmbw	uclmbw
f404	uclmbw	pvs	pvs	uclmbw	pvs	pvs	dlrojo	dlrojo
f405	tal06	dlrojo	ucjela	cillin	dlrojo	tal06	dlrojo	ucjela
f406	rashed	rashed	jcatto	tal06	jcatto	rashed	tal06	tal06
f407	cdomene	cloenarz	wojcik	cdomene	cillin	cillin	cloenarz	jcatto
f408	tal06	rashed	cillin	jcatto	dlrojo	cdomene	dlrojo	tal06
f409	cdomene	ucjela	tal06	ucjela	.....	swr04ojb	shosking	emmaria
f410	swr05vas	.....	.....	swr04ojb	shosking	.....	.....	.....
f411	ndd21	vboppana	cloenarz	uqshe7	uqshe7	hpx0sjw1	hpx00061	uqshe7
f412	meli	uqshe7	hpx0sjw1	cdomene	rashed	uqshe7	hpx0sjw1	uqshe7
f413	ndd21	hpx00061	cloenarz	cdomene	hpx00061	hpx0sjw1	rashed	uqshe7
f414	cdomene	ndd21	vboppana	hpx00061	jw344	cdomene	uqshe7	ndd21
f415	jony	jony	jony	jony	jony	jony	jony	jony
f416	jony	jony	jony	jony	jony	jony	jony	jony
f417	jony	jony	jony	jony	jony	jony	jony	jony
f418	jony	jony	jony	jony	jony	jony	jony	jony
f419	jony	jony	jony	jony	jony	jony	jony	jony
f420	jony	jony	jony	jony	jony	jony	jony	jony
f421	jony	jony	jony	jony	jony	jony	jony	jony
f422	jony	jony	jony	jony	jony	jony	jony	jony

	Total	Alloc	Idle
Batch parallel (capability):	64	64	0
Batch parallel (capacity (S)):	16	15	1
Batch parallel (capacity (L)):	36	36	0
Batch parallel (capacity (vL)):	32	32	0
Batch parallel (development):	12	6	6
Batch parallel (test):	0	0	0
Batch parallel (course):	0	0	0
Interactive shared parallel:	2	0	2
Batch serial:	1	1	0
Unavailable:	0	0	0
Total:	163	154	9

Batch System Concepts cont…

- Common commands include:
  - Submit a job
  - Query status of queues and running jobs
  - Delete a job
- Front-end and back-end nodes may be different processors/architectures
  - Important for compilation, specify required target architecture
- Parallel computing requires job launcher as well as job scheduler
  - Often related

Morar

- Very important on the HPC service - the only way to access the 128 core back-end

- Maximum job size at present: 64 cores

- Uses Sun Grid Engine (SGE) software:
  - use **`qsub`** command to enter jobs
  - use **`qstat`** to monitor the jobs
  - use **`qdel`** to kill jobs

  - **`qmon`** is a gui to give you access to this functionality (and more), if you have x-traffic enabled (e.g. Exceed)

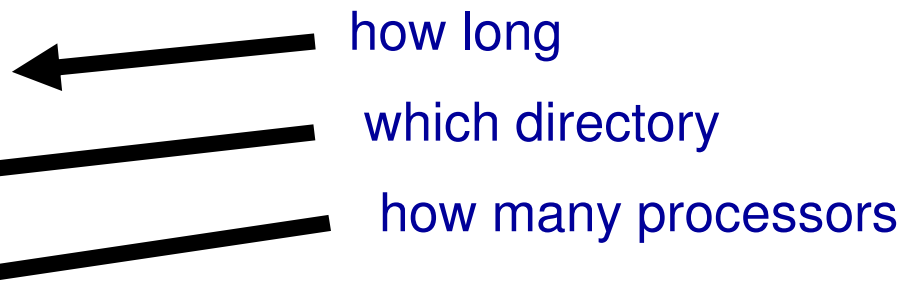
The `qsub` Command

- To enter a job into a queue:
  - **`bash-4.1$ qsub [options] scriptfile`**
- scriptfile contains instructions to run the job
  - **`cd work; echo 'hello'`**
  - starts from your home directory.
- Cannot submit executables directly
  - You always need a UNIX script (can be very simple)
- Jobs will be entered into the queues and a unique jobid returned.

A simple sample script

-
```
#!/bin/bash
#$ -V
#$ -l h_rt=:10:
#$ -cwd
#$ -pe mpi 4

mpiexec -n $NSLOTS ./myprogram
```




Parallel job launcher


Program name

The `qstat` command

- Type **`qstat`** to monitor the queue:

```
[adrianj@ness ~]$ qstat
job-ID prior   name       user      state submit/start at     queue
---------------------------------------------------------------------------
52712  0.55167 ai_mg8.sge user1     r     07/21/2008 15:40:16 8proc_12hours
52730  0.50500 submitEpic user2     r     07/21/2008 16:14:41 16proc_12hours
52774  0.51833 csem.sge   user4     r     07/21/2008 18:01:46 16proc_12hours
52767  0.60500 Myscript   user3     qw    07/21/2008 17:43:22
52713  0.00000 gp_mg8.sge user1     qw    07/21/2008 14:49:54
52714  0.00000 random_mg8 user1     qw    07/21/2008 14:49:54
52735  0.00000 submitEpic user2     qw    07/21/2008 16:23:40
```

The `qdel` command

- To delete a job:
  - Enquire its job-ID with qstat
  - Delete it with qdel (e.g. job-ID: 5789)

```
qdel 5789
```

The `qmon` Command

- To monitor a job or jobs use **`qmon`** which is an X-Windows GUI for Grid Engine - simply type **`qmon`**.

- Select Job Control button (or use pull down menu) to investigate status of jobs (in either Pending, Running or Finished categories

Batch systems

- Required for many “large” machines
- Each system has own syntax and commands
  - Look at manuals etc…
- Remember that most will kill jobs after the maximum time you specified
  - Always check you’ve submitted jobs for long enough
- Different machines setup for different jobs
  - Number of processors and maximum time of queues varies
  - Used to set job policies
- Accounting also often used
  - Job may fail if you’ve not enough time in your budget

Summary

- Lots of different levels to computers
  - Lots of parts as well
- Potential for optimisation if you understand the hardware
- Job of O/S and compilers to do most of this work for you
  - O/S is the interaction point for users
- Systems used by HPC programmers
  - Multi-user
  - Command line based
  - Remotely located
  - Unix or linux based
- Programming fundamentals are the same for all systems
- Understanding what your compilers are doing can be very important for programming and performance
- Always read up on the compilers you will use
- Machine specific libraries usually give better performance
- GNU compilers can let through a lot of bad code
- Performance can be substantially improved using compilers and libraries properly