COURSE REVIEW

COMP4300/8300 PARALLEL SYSTEMS

PROF. JOHN TAYLOR

MAY 2024

Australian National University

Semester 1 SELT is live!

20 May – Survey opens
Check your email or Wattle page for available surveys

21 – 24 May
Food and prizes just for completing your SELT!

Survey runs for 4 weeks
Please provide constructive and respectful feedback *(your teacher can't identify you)*

16 June - Survey closes
IR team perform screening of comments for welfare concerns

8 July
SELT feedback is made available to teachers and course convenors to improve future course delivery

Join us for

free food and prizes

21 – 24 May

11.00am – 2.00pm

CSIT Lab entrance or lawn

(weather dependent)

Australian National University

SELT - Frequently asked questions

What kind of feedback is helpful?

Think about your experience of the course and teaching, and what worked or didn't work for you.

When writing feedback, focus on respectful and constructive language – if you were a teacher, what type of feedback would help you improve the class?

Can teachers see who left specific feedback?

SELT is confidential, and teachers cannot see, or ask to see, the identity of a respondent. Unless you self-identify, for example by using names or describing specific events, teachers cannot identify you.

If your class has very few enrolments, it may be difficult to remain completely anonymous.

Australian National University

Find out more on the *Info for Students* webpage:
https://services.anu.edu.au/learning-teaching/education-data/student-experience-of-learning-teaching-selt/information-for

Assignment 2 & Final Exam

- Assignment 2 is worth 25% of your final mark and is due this Sunday **26/05/2024, 11:55PM**
- Once again, the report has a large relative weight in the final mark, and should be well-written. Submissions without a report will get a very low mark.
- Final Exam will be held on **11/06/2024, 9:00AM-12:15PM, Copland Building G29/G30**
- The Final Exam (FE) is worth 40% of the final mark, however remember that the MSE is redeemable so the mark will be assigned as max(MSE*10/100+FE*40/100, FE*50/100)

Final Exam

- Three-hour pen-and-paper open-book exam with a 15-minute reading time
- Multiple choice questions with content spanning all main course topics – ***25 marks***
- Three sections– ***each worth 25 marks***– with more specific questions requiring more detailed answers and/or problem solving
  - Distributed Memory Parallel Systems Architecture and Programming
  - Shared Memory Parallel Systems Architecture and Programming
  - GPU Architecture and Programming

- Each of last the three sections will contain 3-6 questions on its specific topic

Final Exam Preparations

- All examples in this presentation are taken from past exam papers which are accessible on Wattle.
- Exercising on the past final exams is a very good training for the final exam.
- Of course, the exam will not contain any topic that has not been discussed this year.

Final Exam Preparations – Key topics

- Part I: Classical Parallel Hardware (Flynn Taxonomy, UMA and NUMA Architectures, Distributed- and Shared-Memory Systems)
- Part I: Instruction Level Parallelism (Superscalarity, Pipelining, Out-of-order execution)
- Part I: Dynamic and Static Connectivity Networks
- Part I: Message Passing and MPI (Point-to-Point Communications, Blocking and Non-Blocking Communication, Collectives, Datatypes)
- Part I: Strong and Weak Scaling, Parallel Speedup and Efficiency
- Part I: Network Routing, Communication Cost, Performance Modelling Part I: Synchronous Computations
- Part I: Pipelining and Divide & Conquer Parallelization

Final Exam Preparations – Key Topics

- Part II: Motivation for Shared Memory Parallel Computers (Dennard Scaling, Moore's Law, Limitations of ILP)
- Part II: Process versus Thread Parallelism
- Part II: Pthreads, Thread Memory and Execution Models
- Part II: Thread Synchronization (state diagrams, critical section, semaphores, mutexes & locks)
- Part II: The OpenMP Programming Model
- Part II: Simultaneous Multi-Threading, Single Instruction Multiple Data Part II: GPU Execution Model
- Part II: GPU Memory Model
- Part II: The CUDA Programming Model
- Part II: Snooping-Based Cache Coherence
- Part II: Roofline

Distributed Memory Parallel Systems Architecture and Programming (DMEM)

- Determine performance/cost of DMEM algorithms on a given network topology
- Implement and/or optimize DMEM algorithms using MPI
- Explain semantics of, or amend, a presented MPI code

DMEM: Examples

(b) Explain what the following MPI program is doing, and state the output that is printed if the code is run using 7 MPI processes:

```
int main (int argc, char *argv[]) {
   int token, NP, myrank;
   MPI_Status status;
   MPI_Init (&argc, &argv);
   MPI_Comm_size(MPI_COMM_WORLD, &NP);
   MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
   if (myrank != 0) {
     MPI_Recv(&token, 1, MPI_INT, myrank - 1, 0, MPI_COMM_WORLD, &status);
   } else {
     token = -1;
   }
   token += 2;
   MPI_Send(&token, 1, MPI_INT, (myrank + 1) % NP, 0, MPI_COMM_WORLD);
   if (myrank == 0) {
     MPI_Recv(&token, 1, MPI_INT, NP - 1, 0, MPI_COMM_WORLD, &status);
   }
   printf("rank and token %d %d \n",myrank, token);
   MPI_Finalize();
 }
```

DMEM: Examples

Consider the following algorithm which runs p processors, where p is a power of two and where i (0 <= i < p) is the id of the executing process:

```
int x[m];
...
for (d = p/2; d > 0; d /= 2) {
    if (i % (2*d) == 0)
        send(x, m, i+d);
    else if (i % (2*d) == d)
        recv(x, m, i-d);
}
```

Let t_s be the message startup time, t_w be the transmission cost per unit word, and t_h be the per-hop transmission link latency. Derive expressions for the parallel execution time under a ring topology for both store-forward (SF) routing and cut-through (CT) routing. Give a simple approximate expression for the time reduction ratio of CT over SF, listing any assumptions made.

Shared Memory Parallel Systems Architecture and Programming (SHMEM)

- Implement and/or optimize SHMEM algorithms using OpenMP
- Explain semantics of, or amend, a presented OpenMP code
- Discuss shared memory processor architecture, execution and memory models

SHMEM examples

- Implement and/or optimize SHMEM algorithms using OpenMP
- Explain semantics of, or amend, a presented OpenMP code
- Discuss shared memory processor architecture, execution and memory models

SHMEM examples

(a) Parallelize the following code using openMP pragmas. Assume that the target machine has a cache line size of 128B, that the size of an int is 4B, and the arrays contain ints. Be sure to explicitly specify the "schedule" options that should be used, even if you want to use the default options. For each please rewrite as much code as necessary to make your intent clear. If necessary you can assume that the variable P represents the number of processors to be used. Assume that N is large (in the tens of thousands or more). You must explicitly list all variables within the range of a parallel pragma that are private using the *private()* directive.

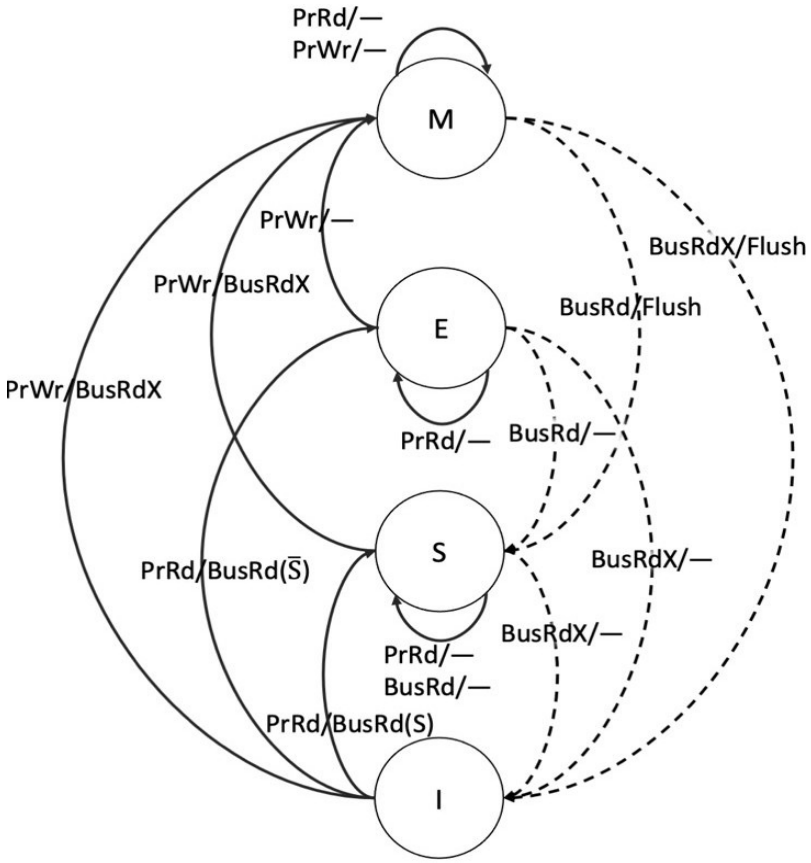
(i)
```
for (i=0;i<N;i++){
      for (j=0;j<N;j++){
        if (A[i*N+j]< B[i*N+j])A[i*N+j]=B[i*N+j];
}}
```

(ii)
```
C[0] = 1;
for (i=1;i<N;i++){
    C[i] = C[i-1];
    for (j=0;j<N;j++){
        C[i] *= A[i*N+j] + B[i*N+j];
}}
```

(iii)
```
typedef struct element
{ int value;
  struct element *next;
} Element;
Element *D[N]; // array of pointers to linked lists of varying length
int C[N];
...
for (i=0;i<N;i++){
    C[i] = computeAverageValueOfAllListElements(D[i]);
}
```

SHMEM examples

processor action	P1 cache miss/hit, X value	P2 cache miss/hit, X value	Bus transaction (processor action initiated)	Bus snooper transaction	C1 state	C2 state	mem[&X]
No-Op	N/A	N/A	–	–	I	I	0
P1Rd X	miss, 0	N/A	BusRd(S")	–			
P1Wr X = 4							
P2Rd X							
P1Wr X = X*2							
P2Wr X = X-3							



GPU Architecture and Programming

- Implement and/or optimize GPU algorithms using CUDA
- Explain semantics of, or amend, a presented CUDA code
- Discuss GPU architecture, execution and memory models

GPU Examples

The following code implements the heat diffusion stencil on an M by N column-major matrix t.

```
iter = 0;
do {
  iter++;
  for (j = 1; j < N-1; j++)
    for (i = 1; i < M-1; i++)
      t1[j*N+i] = 0.25*(t[j*N+i+1] + t[j*N+i-1]+
                        t[(j+1)*N+i]+t[(j-1)*N+i]);
  mxdiff = 0.0;
  for (j = 1; j < N-1; j++)
    for (i = 1; i < M-1; i++) {
      tdiff = fabs((double) (t[j*N+i] - t1[j*N+i]));
      mxdiff = (mxdiff < tdiff)? tdiff: mxdiff;
    }
  for (i = 0; i < M*N; i++) t[i] = t1[i];
} while (mxdiff > converge && iter < Max_iter);
```

Write a CUDA kernel to implement the first loop nest, which will work for any (legal) two-dimensional block and thread sizes. For full marks, it should enable contiguous memory accesses within a thread block, but no further optimizations are required.

GPU Examples

The following code implements the heat diffusion stencil on an M by N column-major matrix t.

```
iter = 0;
do {
  iter++;
  for (j = 1; j < N-1; j++)
    for (i = 1; i < M-1; i++)
      t1[j*N+i] = 0.25*(t[j*N+i+1] + t[j*N+i-1]+
                        t[(j+1)*N+i]+t[(j-1)*N+i]);
  mxdiff = 0.0;
  for (j = 1; j < N-1; j++)
    for (i = 1; i < M-1; i++) {
      tdiff = fabs((double) (t[j*N+i] - t1[j*N+i]));
      mxdiff = (mxdiff < tdiff)? tdiff: mxdiff;
    }
  for (i = 0; i < M*N; i++) t[i] = t1[i];
} while (mxdiff > converge && iter < Max_iter);
```

Describe at least 2 ways how you would optimize your kernel to give better performance, and comment on the expected effectiveness of each way. What would be the main limiting factor to performance in this example?