CSCI 551
Numerical and Parallel Programming

Lecture 4-2 – Parallel Resource Use

February 11, 2025

Exercise #1 Grading

- Grades will be posted before end of week
  - At least 24 hours before Ex #2 due date
  - I make sure I'm not holding more than 1 assignment
- Ideal average for class is 70-80+ on exercises
- Attempt all problems first week, question, improve analysis, verification, and reporting second week
- **Not attempted, incomplete reports are lowest scores**
- **Complete, Clear**, Consistent, Correct!

Assignment #2 Questions?

- Demonstrations of starter code?
- Walk-through of starter code?
- Makefile walk-through?
- Other?

- Using “htop” – what’s going on with my threads?
- Checking memory? (free –h)
- Checking CPU cores? (lscpu or cat /proc/cpuinfo)
- Checking I/O? (iostat and other I/O utilities)

Google & Amazon Interview Question

NxN Transpose & Pivot
CSCI551 C Starter Code
YouTube Solution #1
YouTube Solution #2

Rotate 2D Square Matrix
90-degree rotations
Considered straight-forward

Arbitrary rotation is harder

- Use sine/cosine rotation matrix
- Non-square – requires scaling and/or cropping
- Arbitrary angles has issue of mapping pixels – source to destination X,Y address – collisions and holes
- Must use bi-linear interpolation or similar scheme

Open MP Parallelism: Patterns & Methods

- OpenMP parallel block {}, parallel for loop special case
- OpenMP SPMD threaded function (make function thread aware!)
- OpenMP 3.0+ has task block single/all

- The OpenMP Common Core – Resource
  - https://github.com/tgmattso/OmpCommonCore
  - OmpCommonCore/tree/master/Book/C
  - Book/C/Fig_7.10_fibonacci.c
  - Book/C/Fig_7.11_fibonacciTasks.c

- Compare to csci551/code/hello_openmp/
  - csci551/code/hello_openmp/fib.c
  - csci551/code/hello_openmp/fibsum.c
  - csci551/code/hello_openmp/fibgpt.c (ChatGPT use above!)
  - csci551/code/hello_openmp/tasks.c

- Fibonacci loop-carried dependency
- Pattern is as simple as:
  for(int i = 1; i < N; i++) { A[i] = A[i - 1] + i; }

OmpCommonCore / Book / C / Fig_7.6_taskExp1_C

tgmattso initial commit

Code | Blame | 12 lines (12 loc) · 233 Bytes

```
1   #pragma omp parallel
2   {
3      #pragma omp single
4      {
5         #pragma omp task
6            fred();
7         #pragma omp task
8            daisy();
9         #pragma omp task
10           billy();
11     }  //end of single region
12  } //end of parallel region
```

Thread team

Thread #1

Thread #2

Thread #3

```
sbsiewert@ecc-linux2:~/public_html/csci551/code/hello_openmp$ ./tasks
SINGLE:
I am FRED
I am BILLY
I am DAISY

NOT SINGLE:
I am FRED
I am DAISY
I am DAISY
I am BILLY
I am BILLY
I am FRED
```

Use of single block

No single block

We will tackle loop-carried dependencies in Ex #3!

OpenMP Functional Approach

- OpenMP Pragma – in front of any well-structured block
  - For loops (careful use of collapse, reduction, default, private, shared)
  - { linear code } blocks
  - Function blocks

- Function block OpenMP similar to writing a Pthreads divide & conquer solution
  - Compare piseries.c and piseriesompfunct.c
    - piseries.c is well-structured block pragma
    - piseriesompfunct.c is SPMD or a "function" block design pattern!
  - Function must be thread aware – "I'm thread m of N, so my work is …"
  - Simply place pragma in front of call to function
  - Can simplify reduction too!

SPMD – Parallel Design Pattern

- " … the SPMD pattern is heavily used in message passing programs written in MPI and works well with OpenMP as well. It is probably the most commonly used design pattern … in parallel computing" [1] & [2]

- https://en.wikipedia.org/wiki/Single_program,_multiple_data (special case of MIMD) – functions (program) is thread (worker) aware with specific mapping of work (tasks) and reduction to combine results

- P. 50 in Pacheco – "SPMD programs consist of a single executable that can behave as if it were multiple different programs through the use of conditional branches …"

- SPMD – thread aware functions and rank aware MPI programs or thread aware CUDA kernels

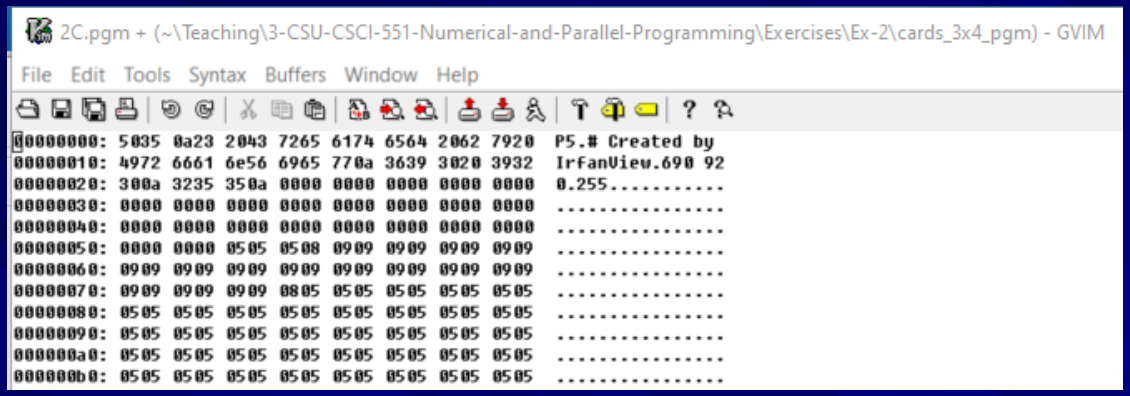
- Pushes beyond simple divide and conquer to map and reduce

[1] Mattson, Timothy G., Yun Helen He, and Alice E. Koniges. "The OpenMP common core." (2019).
[2] Mattson, Timothy G., Beverly Sanders, and Berna Massingill. *Patterns for parallel programming*. Pearson Education, 2004.

Suggestions for Card Rotation - PGM

To simplify, I would recommend using PGM (P5), and not color PPM (P6).

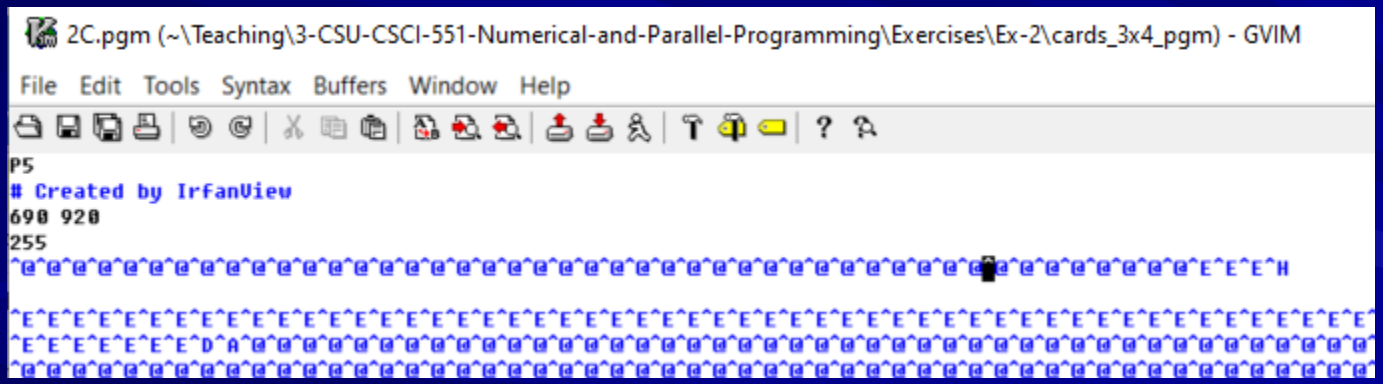
The PGM header shown in “vi” in binary edit mode (“with escape, :%!xxd”):



Fhred and “vi” Binary Mode

- how-to-edit-binary-files-with-vim
- http://frhed.sourceforge.net/en/
- You should not need, but nice to know
- csci551/code/Mat-rotate-with-read-write/

In ASCII mode you will see:



Notes on binary:

- unsigned char
- not char!
- Print as hex
- %X in C
- “hex <<“ in C++

The playing cards are 690x920, so make 1 square array that is 920x920.

The cards are a 3:4 aspect ratio. E.g., 3X230 by 4X230 = 690x920

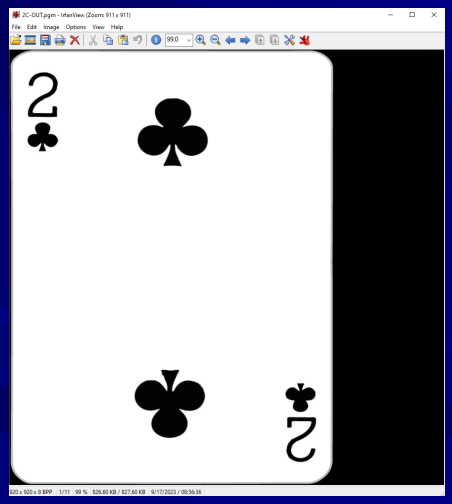
Design Strategy for Card Rotation

1. Write function to read the 690x920 PGM into a 2D array
2. Now write function to write back array with same header
3. Test this – Playing card out should be zero "diff"
4. Assuming #3 works, now write out the whole 920x920 square 2D array
   a. requires updated header dimensions – change to "920 920"
   b. Zero fill unused area (will be black) – this is ok for exercise
5. Test this – looks the same, with black margin – use "eom"
6. Add the code to transpose and pivot 920x920 array
7. Write that out – looks rotated, with black margin – use "eom"
8. Rotate different suits as requested by filename
9. Add parallel directives to rotation code to speed-up
10. Done!

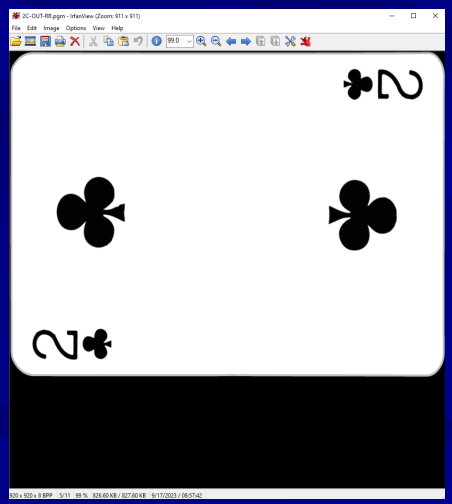
How To Verify Symmetric Card Rotation

- Add columns to make 3:4 (690x920) card 4:4 (920x920)
  - Rotation of a square matrix is simpler than rectangular
  - Can always crop ZERO (black) area later

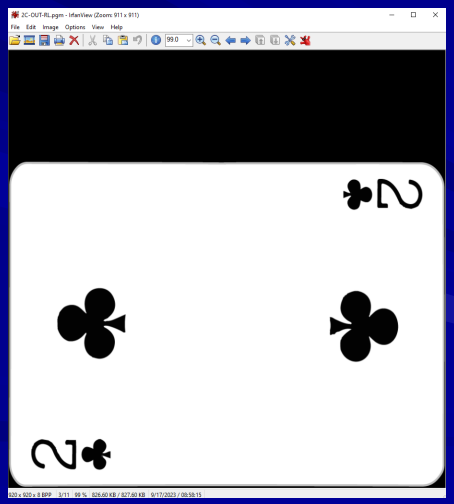
- This also helps with verification for rotation Right or Left
  - Playing cards are symmetric about X, Y axes
  - With ZERO area we see a different between RR and RL



Original Card



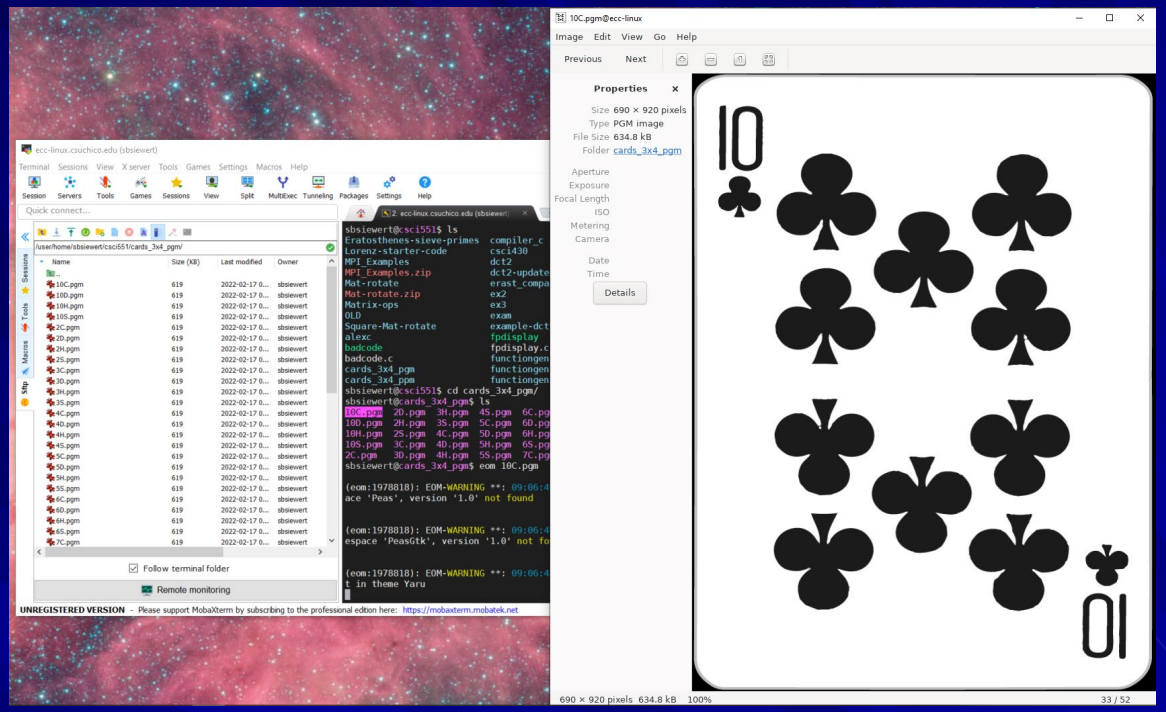
Rotate Right



Rotate Left

A Few More Tips for Card Rotation

- Put the cards in /tmp if you lack disk space on ECC-linux (parallel disks)
- Use “eom” installed on ECC-linux to view them!
- Or user “Irfanview” (Windows download) or GIMP (download)
- MobaXterm automatically does remote display!



Use MobaXterm (Cyberduck) and “eom” for preview

```
sbsiewert@csci551$ cd cards_3x4_pgm/
sbsiewert@cards_3x4_pgm$ ls
10C.pgm  2D.pgm  3H.pgm  4S.pgm  6C.pgm  7D.pgm  8H.pgm  9S.pgm  JC.pgm  KD.pgm  QH.pgm
10D.pgm  2H.pgm  3S.pgm  5C.pgm  6D.pgm  7H.pgm  8S.pgm  AC.pgm  JD.pgm  KH.pgm  QS.pgm
10H.pgm  2S.pgm  4C.pgm  5D.pgm  6H.pgm  7S.pgm  9C.pgm  AD.pgm  JH.pgm  KS.pgm
10S.pgm  3C.pgm  4D.pgm  5H.pgm  6S.pgm  8C.pgm  9D.pgm  AH.pgm  JS.pgm  QC.pgm
2C.pgm   3D.pgm  4H.pgm  5S.pgm  7C.pgm  8D.pgm  9H.pgm  AS.pgm  KC.pgm  QD.pgm
sbsiewert@cards_3x4_pgm$ eom 10C.pgm
```

Extra Help with OpenMP and Pthreads

- Pacheco Chapter 5 – Covers OpenMP
  - assigned reading (syllabus, assignments, notes)
  - Scan and question first, then read, and recite/review before quiz/exam
- Pacheco Chapter 4 – Covers Pthreads
  - assigned reading (syllabus, assignments, notes)
  - Scan and question – read, recite, review for basic Pthread knowledge
- OpenMP – Jaka's Corner on OpenMP, HowTo, OMP Video
- Pthreads – LLNL Pthread Tutorials
- MPI (future) – LLNL MPI Tutorials
- XSEDE – Training Videos, General
- LLNL – Cluster Academy, Internships
- CISL – Internships
- TACC Intro to HPSC
- ORNL – Summer of HPC, Internships, Summer Internships

OpenMP
Generates thread code
Compiler does it!
It can fail silently!
CFLAGS = "-fopenmp"
Focus on loops and functions

Scaling and Speed-Up Concepts

- Scaling is at best linear for "S", the scaling factor, by Amdahl's law (SU=S if P=100%)
  - As S becomes infinite, linear scaling is achieved
  - Start-up & sequential code reduces speed-up
- If S=infinity, then SU depends only on P, SU=1/(1-P), e.g., P=90%, SU=10x
- However, the trick is what is S?
  - Most often # of CPU cores
  - What about SMT ("Hyper-threading") TLP scaling?
  - What about SIMD and Superscalar ILP scaling?
  - What if memory, cache, storage and networking are also scaled at the same time as # of CPU cores?
- Simple view is S=# cores - may under-estimate

Recall Amdahl's Law:

$$\mathrm{SU} = \frac{1}{(1-P)+\frac{P}{S}}$$

$$\mathrm{P}=1.0,\ \mathrm{SU} = \frac{1}{(0)+\frac{1}{S}} = \mathrm{S}$$

$$\mathrm{S}=\infty,\ \mathrm{P}=0.9,\ \mathrm{SU} = \frac{1}{(1-0.9)+0}=10\mathrm{x}$$

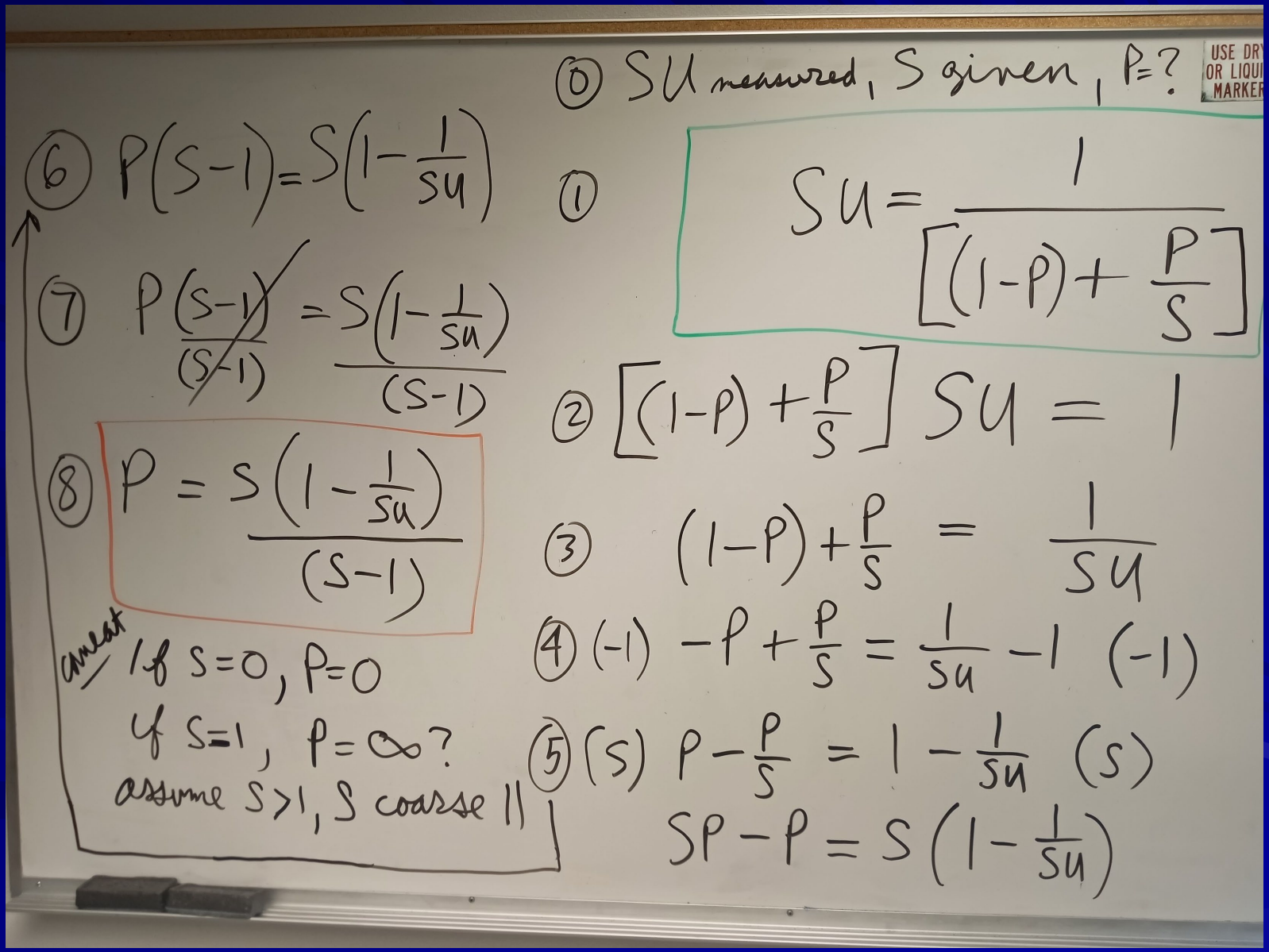
Rearrange Amdahl's Law:

$$\mathrm{SU} = \frac{1}{(1-P)+\frac{P}{S}}\ (\text{measured})$$

[algebraic manipulation]

$$\boldsymbol{P} = \frac{\boldsymbol{S}\left(\boldsymbol{1}-\frac{\boldsymbol{1}}{\boldsymbol{SU}}\right)}{\boldsymbol{S}-\boldsymbol{1}}\ (\text{computed})$$

The Gory Algebra to Isolate "P"



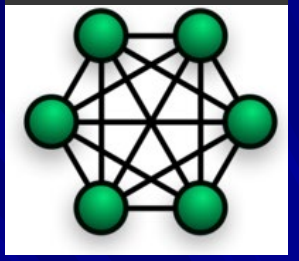
Parallel Scaling – Shared vs. Distributed

- Scale up (shared memory node)
  - Make each cluster node bigger and faster
  - More memory, # of cores, I/O channels, Power
  - UMA and NUMA (Pacheco p. 34)
  - Rack height – 1U, 2U, 4U (U=1.75")
- Scale out (distributed memory – nodes)
  - Pacheco p. 35 – p. 40
  - Increase # of nodes in a cluster
  - Increase speed of interconnection network (1G, 10G, 100G)
    - Latency and bandwidth
  - Increase connectivity in cluster
    - Fully connected mesh
    - Torus (2D, 3D) - HPC
    - General mesh
    - Star
    - Cross-bar (Switch)
    - Bus
    - Ring

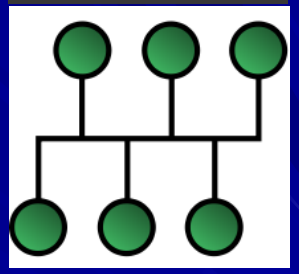
Network_topologies

Mesh @ CSU
- Blocking vs. Non-blocking switches
- 4 port gigE
- 64 port 10GE
- Cost? $10+/port

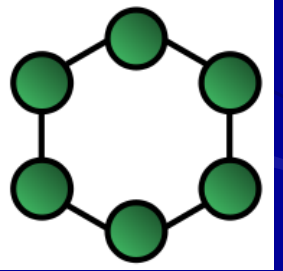
Fully (Strong)



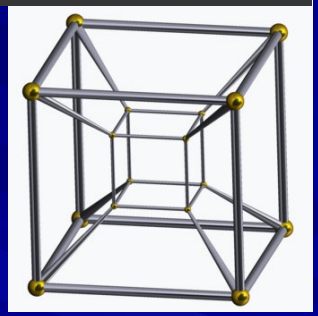
Bus (Shared)



Ring (Simple)



Hypercube



Crossbar - Switch

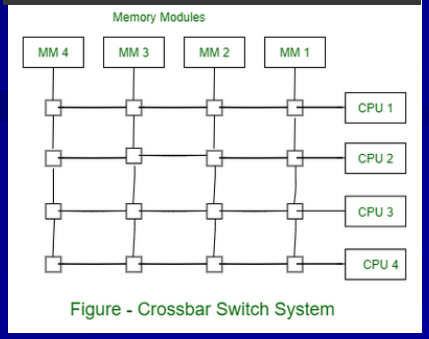
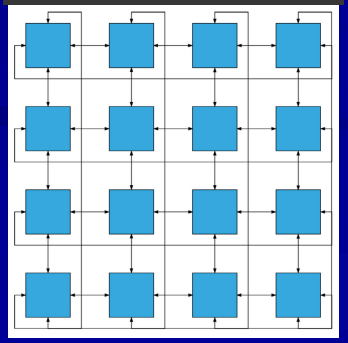


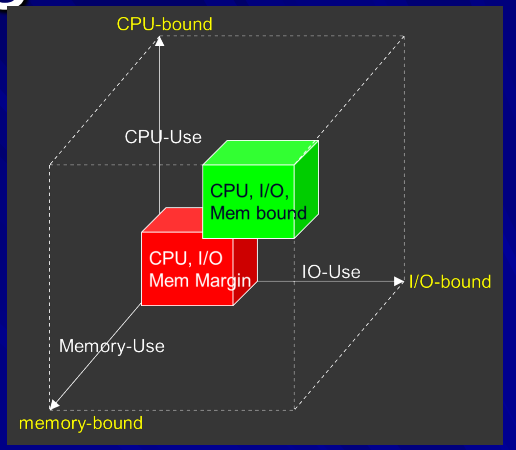
Figure - Crossbar Switch System

2D Torus mesh



HPC (Parallel) Shared Memory Scale Up Limited by Resources

- CPU
  - Clock period
  - Efficiency of each core ALU, FPU, Co-procs
  - # of Cores per CPU
  - Number of CPUs per node
- I/O
  - PCI-E serial byte lane – transfer rate X # of lanes
  - Parallel bus - clock rate X width
- Memory
  - Cache hierarchy and coherency
  - Local Memory
  - Memory scaling (QPI or Hypertransport)
- Power
  - Dynamic Voltage Scaling (Clock speed based on demand)
  - Idle cores (availability based upon need)



Use lshw to record hardware configuration

lsscsi, lsusb, etc.

HPC (Parallel) Limited by Resources

- CPU (Linux CFS – SCHED_OTHER sched default)
  - lscpu
  - Clock period of each core – e.g., 1 nanosecond, 1 GHz
  - Efficiency of each core ALU (Pacheco p. 25 – 28)
    - Pipelined (depth) for CPI=1, IPC=1
    - Vector SIMD
    - Superscalar (instruction multi-issue) for IPC=n, CPI=1/n
    - Hardware SMT (Simultaneous Multi-threading)
      - Direct support in hardware for multiple execution contexts
      - Multi-issue, out of order, retire instructions based on dependencies
  - # of Cores per CPU (UMA shared memory)
  - Number of CPUs per node
    - QPI or Hypertransport scaling (NUMA shared memory)
    - E.g., 4 CPU sockets x 16 cores/CPU = 64 cores/node

HPC (Parallel) Limited by Resources

- I/O (Linux CFQ – fair I/O scheduling)
  - PCI Express Gen1=2.5GT/s, Gen2=5GT/s, Gen3=8GT/s, Gen4=16T/s, …Gen6=64GT/s
  - # of byte lanes = x1, x4, x8, x16, x32, x64
  - sysstat iostat

```
siewerts@siewerts-NUC10i7FNH:~$ iostat -p nvme0n1
Linux 5.13.0-28-generic (siewerts-NUC10i7FNH)   02/14/2022      _x86_64_        (12 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.03    0.00    0.02    0.00    0.00   99.95

Device             tps    kB_read/s    kB_wrtn/s    kB_dscd/s    kB_read    kB_wrtn    kB_dscd
nvme0n1           0.46         2.31         5.69      1724.41    1274140    3133873  949180676
nvme0n1p1         0.00         0.01         0.00         0.94       6547          1     517892
nvme0n1p2         0.46         2.30         5.69      1723.47    1265349    3133872  948662784


siewerts@siewerts-NUC10i7FNH:~$
```

- Bottleneck – always somewhere
  - CPU – for HPC, ideal option
  - I/O – need to scale filesystems and storage
  - Memory – need to scale RAM
  - Network – for MPI, need to scale interconnection (e.g., gigE to 10GE to 100GE)

HPC (Parallel) Limited by Resources

- Memory
  - Large array buffering
  - Buffering multiple arrays over time
  - Buffering (cache) for file I/O
  - free –h, cat /proc/meminfo
- Power
  - Power scaling with scale-up (as nodes upgraded)
  - Power scaling with scale-out (for HPC and data center)
  - Power management
    - Per node – idle cores, unused disk, etc.
    - Per cluster – UPS and PDU (power dist unit)
  - Redundant power – fail over for loss of power

Using Concurrency

- Concurrency for Shared Memory scales processing
  - Using threads in Process address space
  - Shared memory within a process (resource container)
- Concurrency also allows for decoupling of I/O (slow devices) and reduces blocking
  - I/O can be 1000x to 1000000x slower than CPU (nanosec)
  - E.g., disk drive with 2-5 msec access (1000000x slower)
  - E.g., flash with microseconds of access
- Concurrency allows for interactive multi-tasking (reading e-mail while coding)
  - Multi-programming – with multiple apps as processes
  - Most common use

OpenMP and I/O

- OpenMP directives can be used to thread I/O
  - This can speed-up reading and writing files
  - Simpler to read and write explicitly, but does work
- OpenMP should be use for “structured blocks”
  - E.g., for loops (be careful with while and do-while)
  - E.g., functions
  - In theory any block defined with scope of curly braces { block(); }
- OpenMP often fails silently, so add some debug
  - int my_rank = omp_get_thread_num();
  - int thread_count = omp_get_num_threads();
  - printf("Hello from OMP thread %d of %d\n", my_rank, thread_count);

Ex #2 – Rotating Playing Cards

- Reading in and Writing out each Card can be slow
- Use FAST_IO option from PSF starter code
- Read all or up to 52 cards into memory
  - 52 x 690 x 920 bytes = 31.48 Mbytes for PGM
  - 52 x 3 x 690 x 920 bytes = 94.44 Mbytes for PPM

```
sbsiewert@~$ hostname
o251-14
sbsiewert@~$ free -h
               total        used        free      shared  buff/cache   available
Mem:           7.7Gi       605Mi       4.3Gi        94Mi       2.8Gi       6.5Gi
Swap:          4.0Gi          0B       4.0Gi
sbsiewert@~$
```

- Threads can process cards in memory as soon as they are read in (marked ready)
- Start writing cards back out as soon as rotated (marked done)
- PGM is simpler than PPM

Advanced Topics on I/O

If Linux I/O is slow, it can be threaded and pipelined

Linux Concepts for Resources

- CFS – Completely Fair Scheduler (Processing)
  - SMP load balancing
  - Nobody starves!
  - Some processes get resources more often (nice levels)
  - https://docs.kernel.org/scheduler/
- BFQ & CFQ – Completely Fair Queue (I/O)
  - I/O should be fair, just like processing
  - You can set the policy
- Memory is allocated to Processes (Memory)
  - Address space (contains memory references)
  - Keeps your process safe from mine, and vice versa
  - Sharing between processes requires shm operations, files, or message passing!

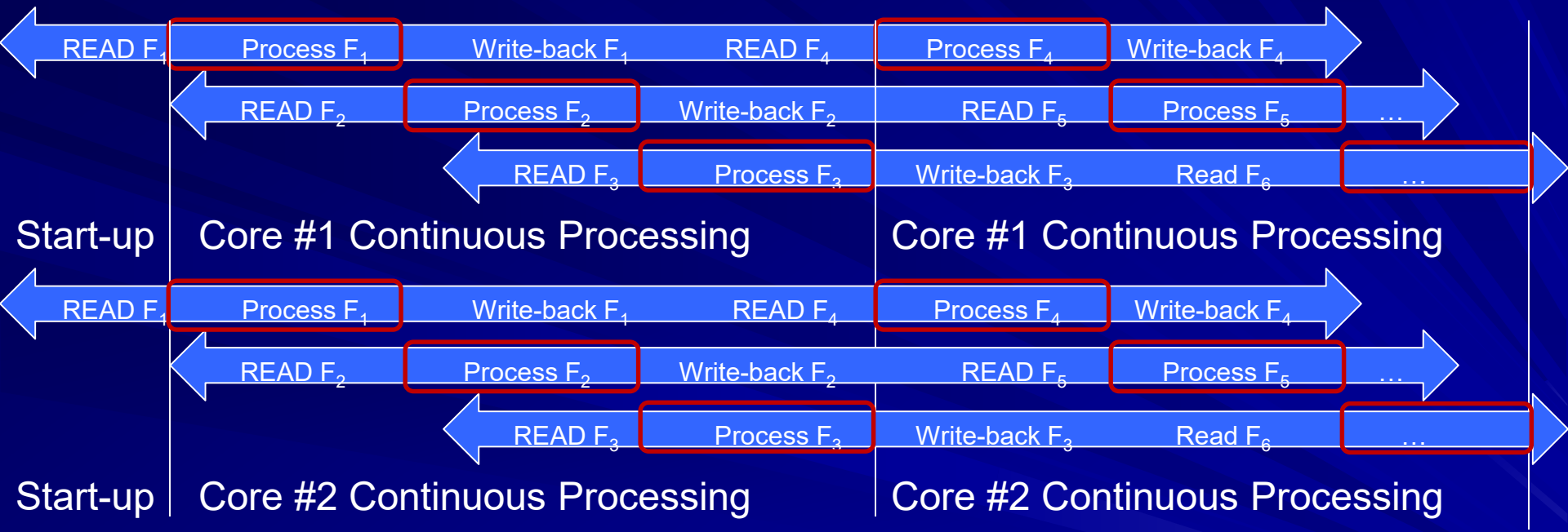
Hiding I/O Latency – Overlapping with Processing

- Simple Design – Each Thread has READ, PROCESS, WRITE-BACK Execution

- Frame rate is READ+PROCESS+WRITE latency – e.g. 10 fps for 100 milliseconds
  - If READ is 70 msec, PROCESS is 10 msec, and WRITE-BACK 20 msec, predominate time is I/O time, not processing
  - Disk drive with 100 MB/sec READ rate can only read 16 fps, 62.5 msec READ latency

Hiding I/O Latency (software pipeline)

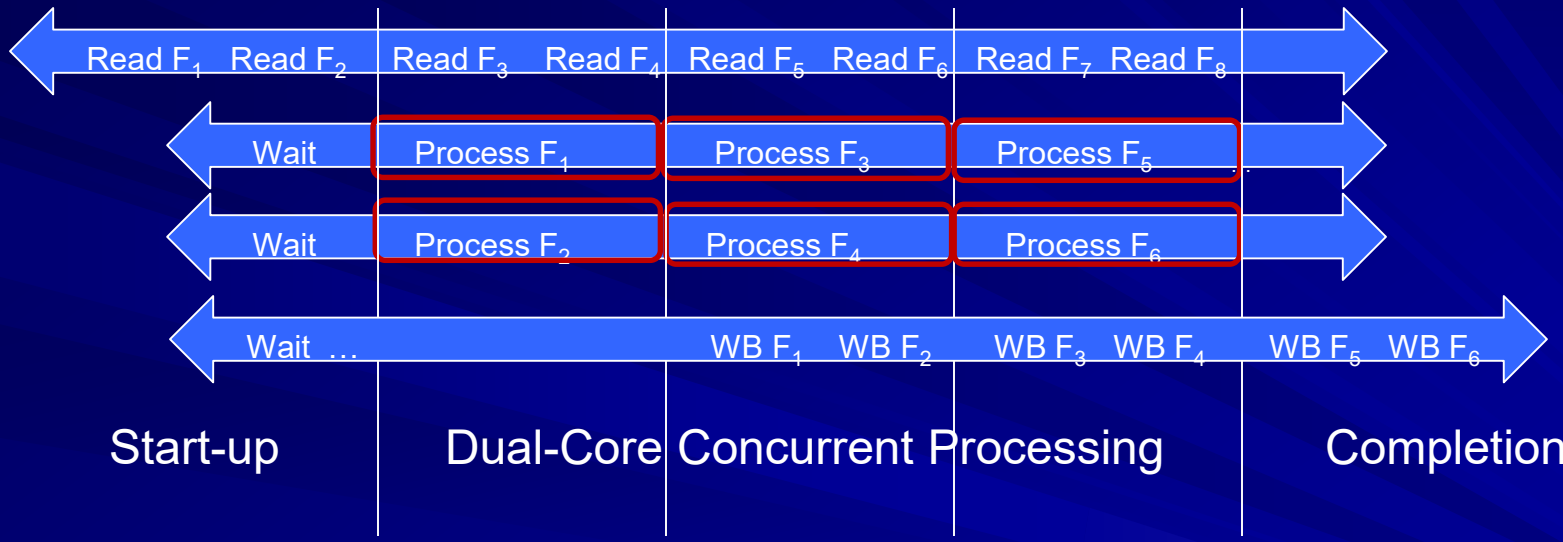
- Schedule Multiple Overlapping Threads?



- Requires $N_{threads} = N_{stages} \times N_{cores}$
- 1.5 to 2x Number of Threads for SMT (Hyper-threading)
- For IO Stage Duration Similar to Processing Time
- More Threads if IO Time (Read+WB+Read) >> 3 x Processing Time

Hiding Latency – Dedicated I/O

- Schedule Reads Ahead of Processing



- Requires $N_{threads} = 2 + N_{cores}$
- Synchronize Frame Ready/Write-backs
- Balance Stage Read/Write-Back Latency to Processing
- 1.5 to 2x Threads for SMT (Hyper-threading)

Tips for I/O Scheduling

- blockdev --getra /dev/sda
  - Should return 256
  - Means that reads read-ahead up to 128K
  - Function calls – read, fread should request as much as possible
  - Check “actual bytes read”, re-read as needed in a loop
- blockdev --setra /dev/sda 16384 (8MB)

- Switch CFQ to Deadline
  - Use “lsscsi” to verify your disk is /dev/sda … substitute block driver interface used for file system if not sda
  - cat /sys/block/sda/queue/scheduler
  - echo deadline > /sys/block/sda/queue/scheduler
- Options are noop, cfq, deadline

20+ Tips to Speed-up Linux Apps Processing, I/O, Storage

As a user:

1. Write efficient code in thread loops (avoid printf, unnecessary system calls, memcpy, etc.)
2. Measure cost of system and API calls (with timestamps or profiling)
3. Always pre-create all threads and request service via binary semaphore or message queue
4. Turn off application graphics and visualization for test and demonstration
5. De-couple storage I/O from processing with a ring buffer and I/O threads
6. Use a RAM disk for temporary file
7. Oversubscribe cores with threads – at least 2x threads per core (don't leave them idle)
8. Overlap I/O with multiple services and ring-buffers (with read-ahead and write-back queues)
9. Avoid "memcpy" and instead, try zero-copy schemes such as heap message queues or ring buffers with pointers to a pre-allocated heap
10. Avoid malloc in service loops – try to pre-allocate
11. Use sysprof, gprof, and syslog to figure out where you are bottlenecked
12. Turn on NEON or SSE SIMD compile options if available – for frame transformation speed-up (-mcpu=cortex-a7 -mfpu=neon-vfpv4 – look up latest switches for R-Pi and Intel)
13. Compile with higher levels of optimization ("-O3" for example)
14. Don't use printf for performance runs (use syslog or trace-cmd or other tracing)
15. Simplify syslogs and use CSV output format
16. Use efficient I/O interfaces with Zero Copy (e.g. Linux MMAP)

As root:

1. Use irqbalance, if possible, to distribute interrupts - https://linux.die.net/man/1/irqbalance
2. Turn off system graphics and visualization for test and demonstration
3. Tune parameters like CFQ and setra if possible (tips in these notes)
4. Use "mlock", if possible, to lock down pages allocated with a pre-malloc
5. Set priority of process with nice - -n (dash minus level)