COMP 322: Fundamentals of Parallel Programming

Lecture 1: The What and Why of Parallel Programming; Task Creation & Termination (async, finish)

Vivek Sarkar

Department of Computer Science, Rice University

vsarkar@rice.edu

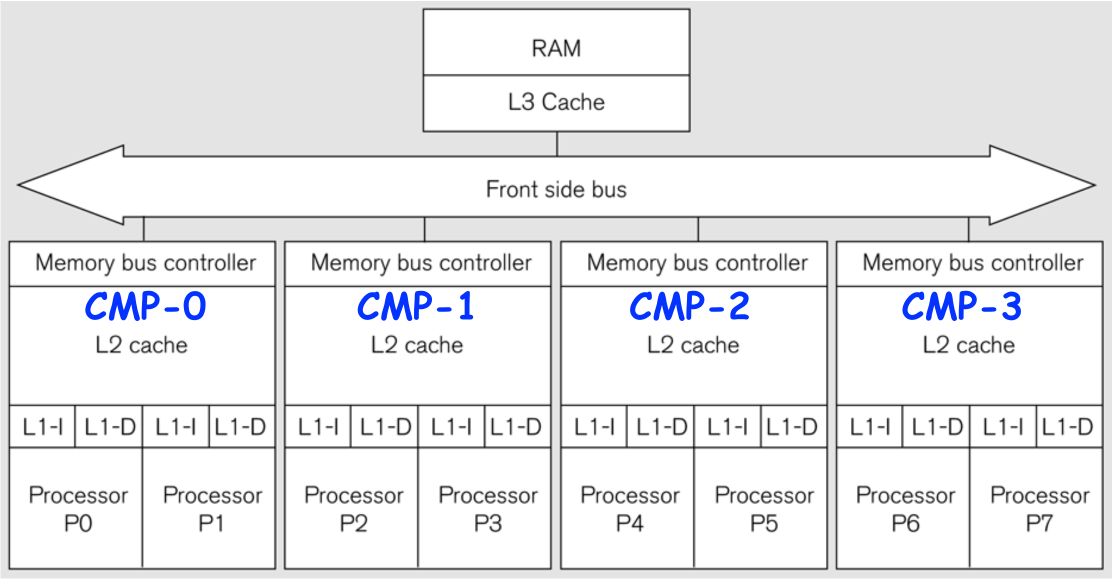
https://wiki.rice.edu/confluence/display/PARPROG/COMP322

Acknowledgments

- CS 194 course on “Parallel Programming for Multicore” taught by Prof. Kathy Yelick, UC Berkeley, Fall 2007
  - http://www.cs.berkeley.edu/~yelick/cs194f07/
- “Principles of Parallel Programming”, Calvin Lin & Lawrence Snyder, Addison-Wesley 2009
- COMP 322 Module 1 handout, Sections 0.1, 0.2, 1.1
- edX lecture and demonstration videos for Module 1, topic 1.1

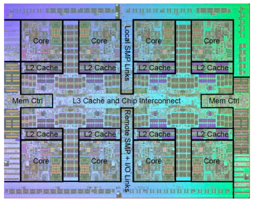
What is Parallel Computing?

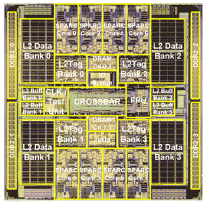
- Parallel computing: using multiple processors in parallel to solve problems more quickly than with a single processor and/or with less energy
- Examples of a parallel computer
  - An 8-core Symmetric Multi-Processor (SMP) consisting of four dual-core Chip Multi-Processors (CMPs)

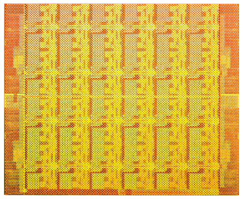


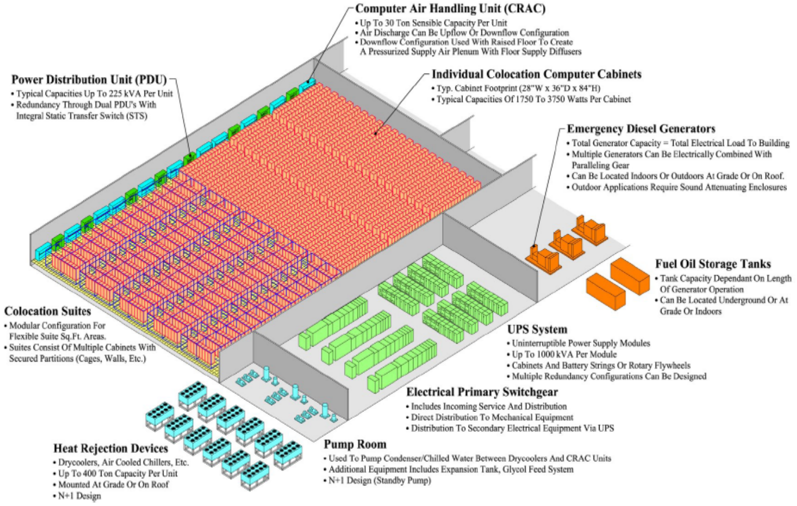
Source: Figure 1.5 of Lin & Snyder book, Addison-Wesley, 2009

All Computers are Parallel Computers --- Why?









Computer Air Handling Unit (CRAC)
- Up To 30 Ton Sensible Capacity Per Unit
- Air Discharge Can Be Upflow Or Downflow Configuration
- Downflow Configuration Used With Raised Floor To Create A Pressurized Supply Air Plenum With Floor Supply Diffusers

Power Distribution Unit (PDU)
- Typical Capacities Up To 225 kVA Per Unit
- Redundancy Through Dual PDU's With Integral Static Transfer Switch (STS)

Individual Colocation Computer Cabinets
- Typ. Cabinet Footprint (28"W x 36"D x 84"H)
- Typical Capacities Of 1750 To 3750 Watts Per Cabinet

Emergency Diesel Generators
- Total Generator Capacity = Total Electrical Load To Building
- Multiple Generators Can Be Electrically Combined With Paralleling Gear
- Can Be Located Indoors Or Outdoors At Grade Or On Roof.
- Outdoor Applications Require Sound Attenuating Enclosures

Fuel Oil Storage Tanks
- Tank Capacity Dependant On Length Of Generator Operation
- Can Be Located Underground Or At Grade Or Indoors

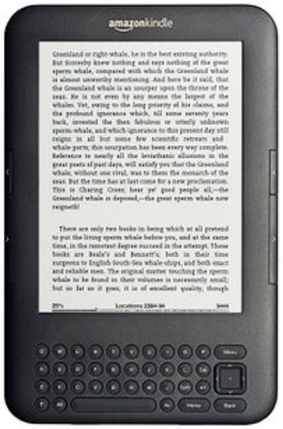
Colocation Suites
- Modular Configuration For Flexible Suite Sq.Ft. Areas.
- Suites Consist Of Multiple Cabinets With Secured Partitions (Cages, Walls, Etc.)

UPS System
- Uninterruptible Power Supply Modules
- Up To 1000 kVA Per Module
- Cabinets And Battery Strings Or Rotary Flywheels
- Multiple Redundancy Configurations Can Be Designed

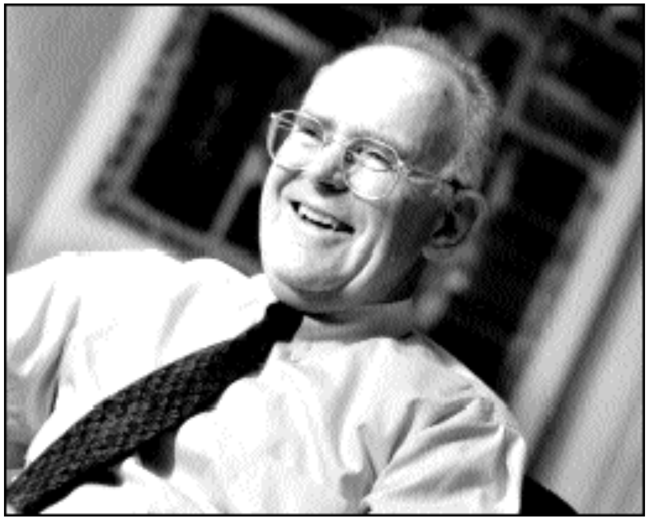
Electrical Primary Switchgear
- Includes Incoming Service And Distribution
- Direct Distribution To Mechanical Equipment
- Distribution To Secondary Electrical Equipment Via UPS

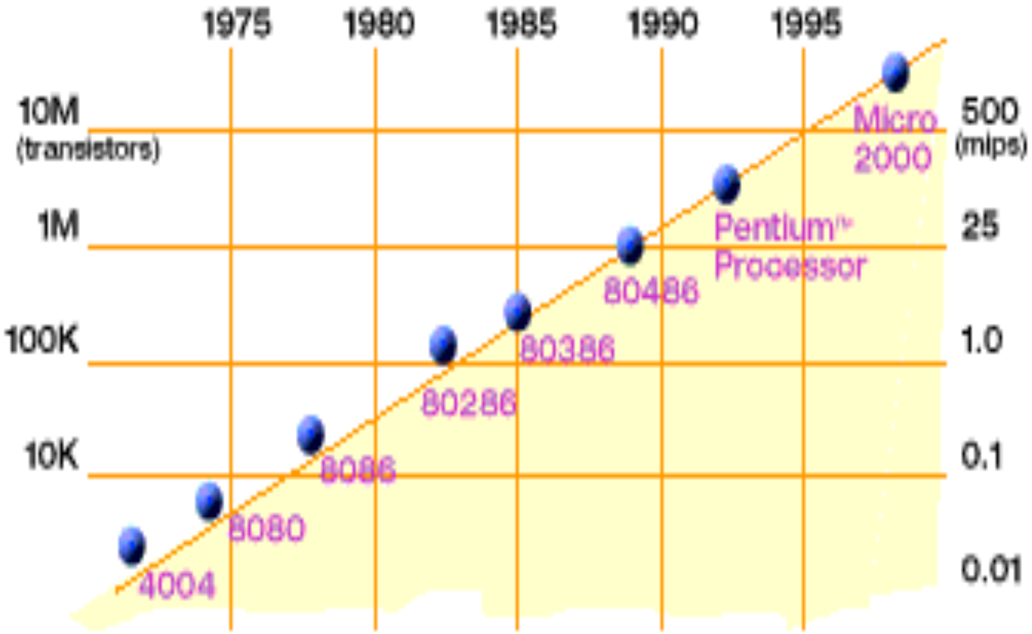
Heat Rejection Devices
- Drycoolers, Air Cooled Chillers, Etc.
- Up To 400 Ton Capacity Per Unit
- Mounted At Grade Or On Roof
- N+1 Design

Pump Room
- Used To Pump Condenser/Chilled Water Between Drycoolers And CRAC Units
- Additional Equipment Includes Expansion Tank, Glycol Feed System
- N+1 Design (Standby Pump)



Moore's Law





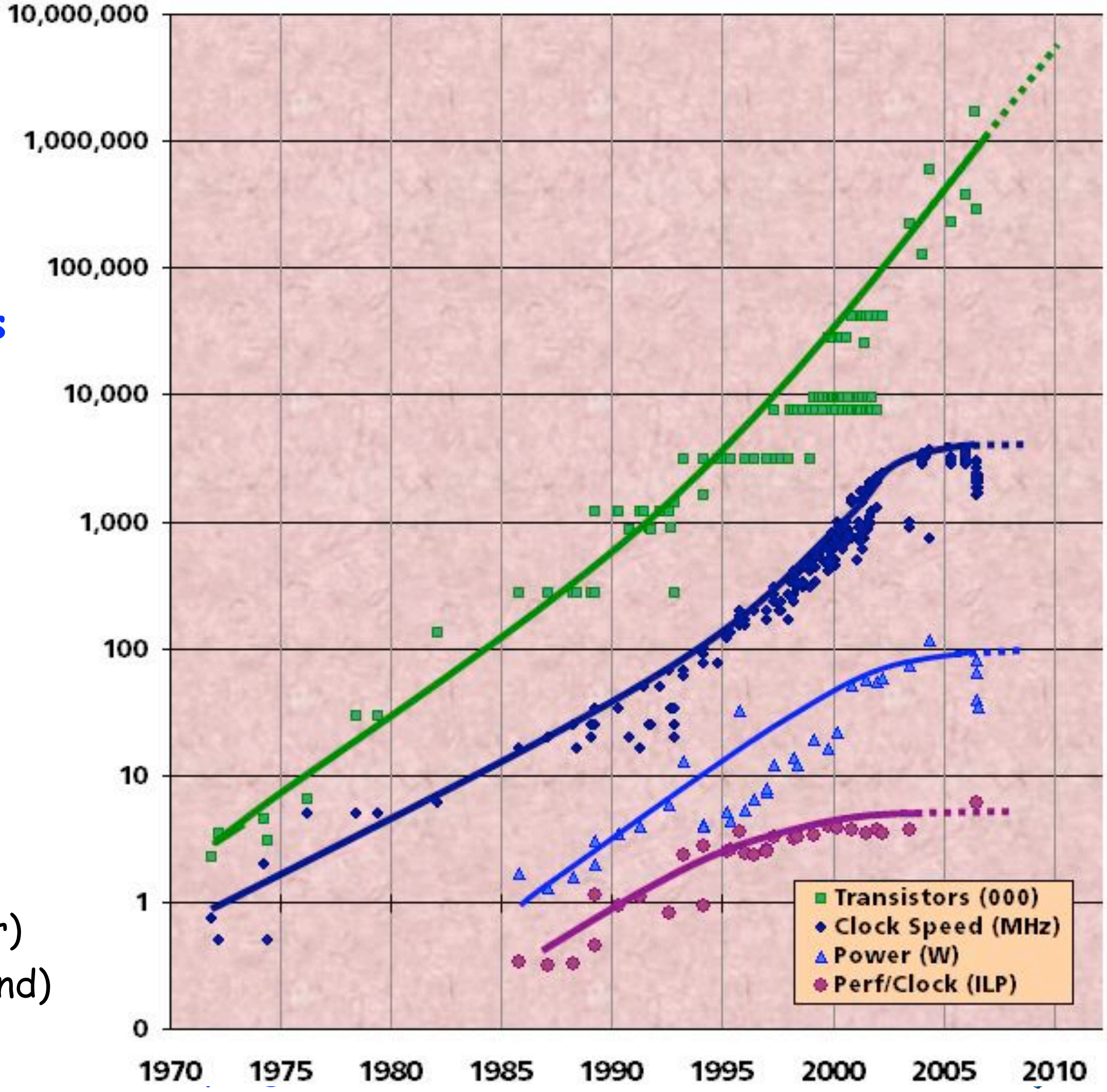
Gordon Moore (co-founder of Intel) predicted in 1965 that the transistor density of semiconductor chips would double roughly every 1-2 years

Slide source: Jack Dongarra

Resulted in CPU clock speed doubling roughly every 18 months, but not any longer

Current Technology Trends

- Chip density is continuing to increase ~2x every 2 years
  - — Clock speed is not
  - — Number of processors is doubling instead
- Parallelism must be managed by software



Source: Intel, Microsoft (Sutter) and Stanford (Olukotun, Hammond)

Parallelism Saves Power (Simplified Analysis)

Power = (Capacitance) * (Voltage)^2 * (Frequency)

➔ Power is proportional to (Frequency)^3

<u>Baseline example</u>: single 1GHz core with power P

<u>Option A</u>: Increase clock frequency to 2GHz ➔ Power = 8P

<u>Option B</u>: Use 2 cores at 1 GHz each ➔ Power = 2P

- Option B delivers same performance as Option A with 4x less power … provided software can be decomposed to run in parallel!

A Real World Example

- Fermi vs. Kepler GPU chips from NVIDIA's GeForce 600 Series
  - Source: http://www.theregister.co.uk/2012/05/15/nvidia_kepler_tesla_gpu_revealed/

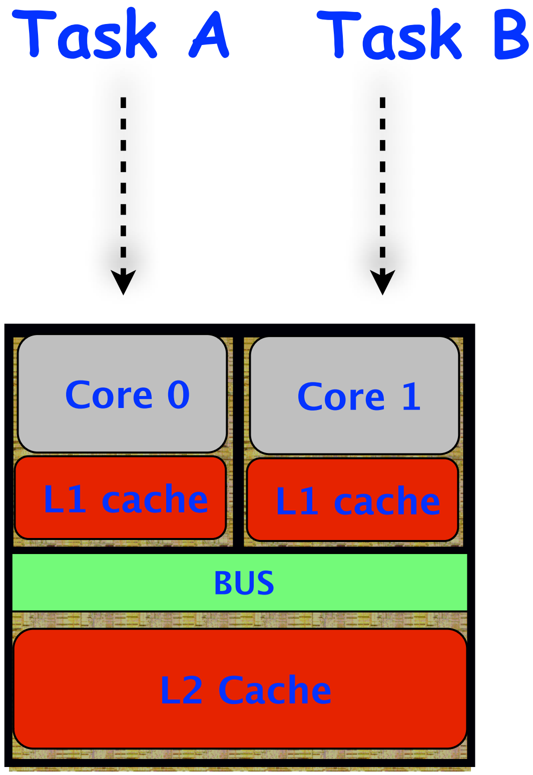
	Fermi chip (released in 2010)	Kepler chip (released in 2012)
Number of cores	512	1,536
Clock frequency	1.3 GHz	1.0 GHz
Power	250 Watts	195 Watts
Peak double precision floating point performance	665 Gigaflops	1310 Gigaflops (1.31 Teraflops)

Scope of Course

- Fundamentals of parallel programming
  - Primitive constructs for task creation & termination, collective & point-to-point synchronization, task and data distribution, and data parallelism
  - Abstract models of parallel computations and computation graphs
  - Parallel algorithms & data structures including lists, trees, graphs, matrices
  - Common parallel programming patterns
- Habanero-Java (HJ) library as a pedagogic parallel programming model
  - Developed in the Habanero Multicore Software Research project at Rice
- Java Concurrency
- Beyond HJ and Java: Map-Reduce, CUDA, MPI
- Weekly quizzes in edX (due by Friday for each week)
- 6 Homeworks with written assignments and programming assignments
  - Abstract metrics
  - Real parallel systems in Rice's Data Center

What is Parallel Programming?

- Specification of operations that can be executed in parallel
- A parallel program is decomposed into sequential subcomputations called <u>tasks</u>
- Parallel programming constructs define task creation, termination, and interaction



Schematic of a dual-core Processor

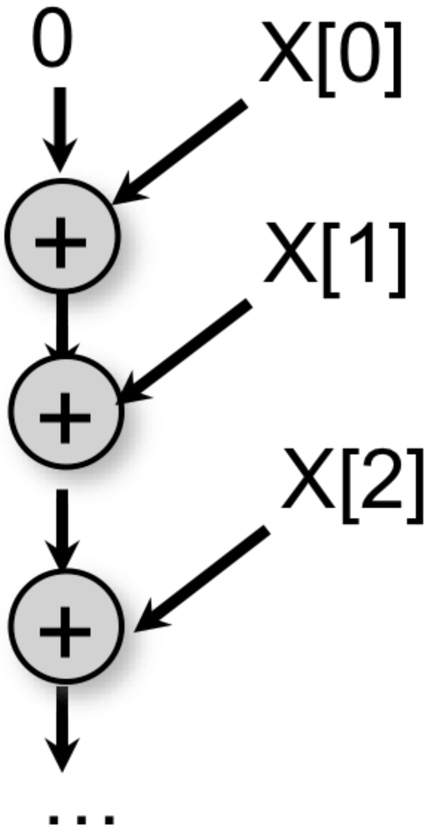
Example of a Sequential Program: Computing the sum of array elements

```
int sum = 0;
for (int i=0 ; i < X.length ; i++)
    sum += X[i];
```

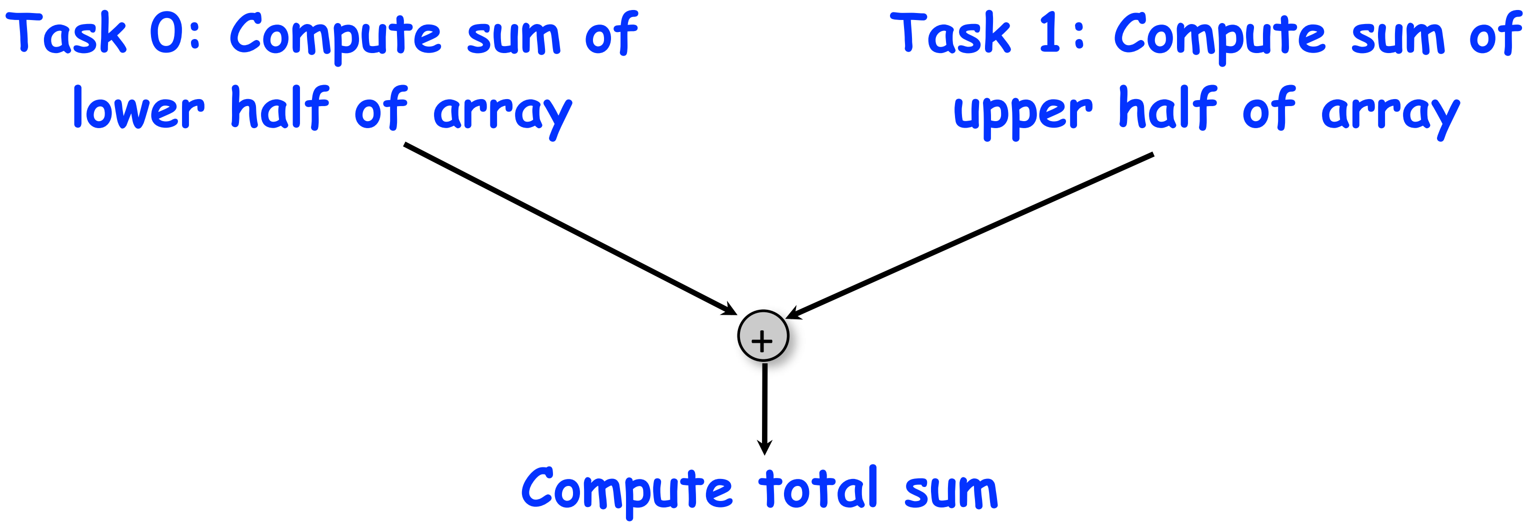
Observations:

- The decision to sum up the elements from left to right was arbitrary
- The computation graph shows that all operations must be executed sequentially

Computation Graph



Parallelization Strategy for two cores (Two-way Parallel Array Sum)



Task 0: Compute sum of lower half of array

Task 1: Compute sum of upper half of array

Compute total sum

Basic idea:

- Decompose problem into two tasks for partial sums
- Combine results to obtain final answer
- Parallel divide-and-conquer pattern

Async and Finish Statements for Task Creation and Termination

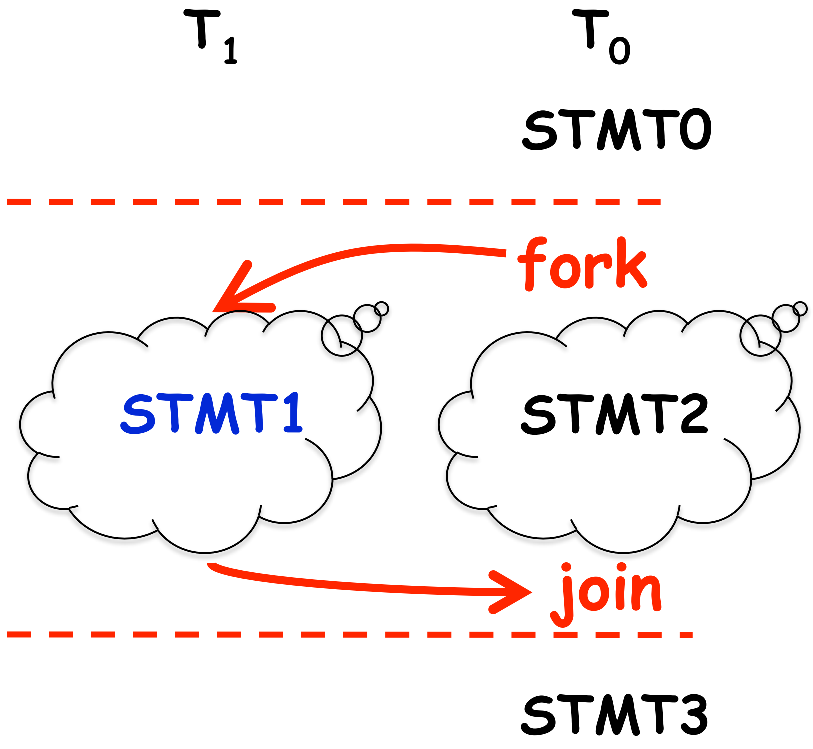
async S

- Creates a new child task that executes statement S

finish S

- Execute S, but wait until *all* asyncs in S's scope have terminated.

```
// T0(Parent task)
STMT0;
finish {     //Begin finish
  async {
    STMT1; //T1(Child task)
  }
  STMT2;     //Continue in T0
             //Wait for T1
}            //End finish
STMT3;       //Continue in T0
```



T_1 T_0

STMT0

fork

STMT1

STMT2

join

STMT3

Two-way Parallel Array Sum using async & finish constructs

```
// Start of Task T0 (main program)
sum1 = 0; sum2 = 0; // sum1 & sum2 are static fields
finish {
  async // Child task computes sum of lower half of array
    for(int i=0; i < X.length/2; i++) sum1 += X[i];
  // Parent task computes sum of upper half of array
  for(int i=X.length/2; i < X.length; i++) sum2 += X[i];
}
// Parent task waits for child task to complete (join)
return sum1 + sum2;
```

NOTE: This example uses pseudocode notation for async & finish instead of concrete Habanero Java library syntax

Lecture Modules

1. Deterministic Shared-Memory Parallelism: creation and coordination of parallelism, collective & point-to-point synchronization (phasers, barriers), abstract performance metrics (work, span, critical paths), Amdahl's Law, weak vs. strong scaling, data races and determinism, data race avoidance (immutability, futures, accumulators, dataflow), deadlock avoidance, abstract vs. real performance (granularity, scalability), parallel sorting algorithms.

2. Nondeterministic Shared-Memory Parallelism and Concurrency: critical sections, atomicity, isolation, high level data races, nondeterminism, linearizability, liveness/progress guarantees, actors, request-response parallelism

3. Distributed-Memory Parallelism and Locality: memory hierarchies, cache affinity, false sharing, message-passing (MPI), communication overheads (bandwidth, latency), MapReduce, systolic arrays, accelerators, GPGPUs.

COMP 322 Course Information: Spring 2014

- **IMPORTANT:**
  - Send email to comp322-staff@mailman.rice.edu if you did NOT receive a welcome email from us
  - Bring your laptop to this week's lab (Section A01: Monday, Section A02: Wednesday) to ensure that you are properly set up with all class infrastructure
- **Course Requirements:**

—Homeworks (6)	40%	(written + programming components)
—Exams (2)	40%	(take-home written exams)
—Weekly Quizzes	10%	(on EdX)
—Class Participation	10%	(in-class activities, lab attendance, classroom & on-line discussions, Q&A)

- **HW1 will be assigned on Jan 17th and be due on Jan 31st**