Parallel Programming Concepts
What is concurrency?

- **What is a sequential program?**
  - A single thread of control that executes one instruction and when it is finished execute the next logical instruction

- **What is a concurrent program?**
  - A collection of autonomous sequential threads, executing (logically) in parallel

- **The implementation (i.e. execution) of a collection of threads can be:**
  - **Multiprogramming**
    - Threads multiplex their executions on a single processor.
  - **Multiprocessing**
    - Threads multiplex their executions on a multiprocessor or a multicore system
  - **Distributed Processing**
    - Processes multiplex their executions on several different machines
Concurrency and Parallelism

- Concurrency is not (only) parallelism

- Interleaved Concurrency
  - Logically simultaneous processing
  - Interleaved execution on a single processor

- Parallelism
  - Physically simultaneous processing
  - Requires a multiprocessor or a multicore system
Example Parallelization

- **Data parallel**
  - Perform same computation but operate on different data

- **A single process can fork multiple concurrent threads**
  - Each thread encapsulates its own execution path
  - Each thread has local state and shared resources
  - Threads communicate through shared resources such as global memory

```c
for (i = 0; i < 12; i++)
    C[i] = A[i] + B[i];
```
Types of Parallelism

- **Data parallelism**
  - Perform same computation but operate on different data

- **Control (task) parallelism**
  - Perform different functions

```c
pthread_create(/* thread id */, /* attributes */, /* any function */, /* args to function */);
```
Understanding Performance

- What factors affect performance of parallel programs?
  - **Coverage** or extent of parallelism in algorithm
  - **Granularity** of partitioning among processors
  - **Locality** of computation and communication
Limits to Performance Scalability

- Not all programs are “embarrassingly” parallel

- Programs have sequential parts and parallel parts

Sequential part (data dependence)

Parallel part (no data dependence)

```
a = b + c;
d = a + 1;
e = d + a;
for (i=0; i < e; i++)
    M[i] = 1;
```
Coverage

- **Amdahl's Law**: The performance improvement to be gained from using some faster mode of execution is limited by the fraction of the time the faster mode can be used.
  - Demonstration of the law of diminishing returns
Performance Scalability

Typical speedup is less than linear
Understanding Performance

- **Coverage** or extent of parallelism in algorithm
- **Granularity** of partitioning among processors
- **Locality** of computation and communication
Granularity

- Granularity is a qualitative measure of the ratio of computation to communication

- Computation stages are typically separated from periods of communication by synchronization events
Fine vs. Coarse Granularity

- **Fine-grain Parallelism**
  - Low computation to communication ratio
  - Small amounts of computational work between communication stages
  - Less opportunity for performance enhancement
  - High communication overhead

- **Coarse-grain Parallelism**
  - High computation to communication ratio
  - Large amounts of computational work between communication events
  - More opportunity for performance increase
  - Harder to load balance efficiently
Granularity

- Parallel loops/regions have overhead
  - Invoking the parallel loops
  - Executing the barriers
  - Cache and synchronization effects
  - Thread management
- If the coverage is perfect, but the program invokes a very large number of very small parallel loops, then performance might be limited by granularity
Granularity

- Time in cycles for empty parallel do

- One should not parallelize a loop or region unless it takes significantly more time to execute than the parallel overhead

<table>
<thead>
<tr>
<th>Processors</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1800</td>
</tr>
<tr>
<td>2</td>
<td>2400</td>
</tr>
<tr>
<td>4</td>
<td>2900</td>
</tr>
<tr>
<td>8</td>
<td>4000</td>
</tr>
<tr>
<td>16</td>
<td>8000</td>
</tr>
</tbody>
</table>
The Load Balancing Problem

- Processors that finish early have to wait for the processor with the largest amount of work to complete
  - Leads to idle time, lowers utilization

```c
// PPU tells all SPEs to start
for (int i = 0; i < n; i++) {
    spe_write_in_mbox(id[i], <message>);
}

// PPU waits for SPEs to send completion message
for (int i = 0; i < n; i++) {
    while (spe_stat_out_mbox(id[i]) == 0);
    spe_read_out_mbox(id[i]);
}
```
Static Load Balancing

- Programmer make decisions and assigns a fixed amount of work to each processing core a priori
  
- Works well for homogeneous multicores
  - All core are the same
  - Each core has an equal amount of work

- Not so well for heterogeneous multicores
  - Some cores may be faster than others
  - Work distribution is uneven
Dynamic Load Balancing

● When one core finishes its allocated work, it takes on work from core with the heaviest workload

● Ideal for codes where work is uneven, and in heterogeneous multicore
Granularity and Performance Tradeoffs

1. Load balancing
   - How well is work distributed among cores?

2. Synchronization
   - Are there ordering constraints on execution?
Data Dependence Graph
Dependence and Synchronization

Synchronisation Points

P1 → P2
P2 → P3
P3 → P3
P3 → P3
P3 → P3
Synchronisation Removal

Synchronisation Points

P1
P2
P3
P3
P3
P3
Granularity and Performance Tradeoffs

1. Load balancing
   - How well is work distributed among cores?

2. Synchronization
   - Are there ordering constraints on execution?

3. Communication
   - Communication is not cheap!
Understanding Performance

- **Coverage** or extent of parallelism in algorithm
- **Granularity** of data partitioning among processors
- **Locality** of computation and communication
Locality of Memory Accesses (Shared Memory)

```c
for (i = 0; i < 16; i++)
    C[i] = A[i] + ...;
```
Locality of Memory Accesses (Shared Memory)

```c
for (i = 0; i < 16; i++)
    C[i] = A[i] + ...;
```

fork (threads)

join (barrier)

memory banks


memory interface
Summary of Parallel Performance Factors

- Coverage or extent of parallelism in algorithm
- Granularity of data partitioning among processors
- Locality of computation and communication

- … so how do I parallelize my program?