Programming with CUDA
and Parallel Algorithms

Waqar Saleem
Jens Müller
Recap

- Writing OpenCL programs
Today

• Analyzing parallel algorithms
• PRAM: EREW, CREW, ERCW, CRCW
• Definitions: cost, speed up, efficiency
• Amdahl’s law
• Brent’s theorem
• Project introductions
A Common Model for Parallel Computation

• Diverse parallel architectures exist
  • A common model will necessarily oversimplify

• Forcing any one model will cause giving up benefits of other models

• Analysis for one model might be completely invalid on another model
Parallel Random Access Machine (PRAM)

• A PRAM is a “shared memory abstract machine” (wikipedia)

• Synchronization and communication costs are not considered
PRAM Models

- Restrict memory access to resolve conflicts
- EREW - Exclusive Read Exclusive Write
- CREW - Concurrent Read Exclusive Write
- ERCW - Exclusive Read Concurrent Write
- CRCW - Concurrent Read Concurrent Write

- Concurrent writes can be
  - Common - All write the same value, otherwise illegal
  - Arbitrary - An arbitrary write succeeds, others fail
  - Priority - Writes are prioritized
  - Associative - All values to be written are reduced to a single value
PRAM Algorithms

- Potentially infinite processors
- Potentially infinite memory
- Random Access memory
- MIMD
  - processors are in lock-step synchronization: they execute instructions together
  - these instructions may vary per processor
- No resource contention, except memory
PRAM instruction

• PRAM instructions proceed in three steps
  • Read from shared memory
  • Compute in private memory
  • Store in shared memory
• Each instruction takes one time unit
• An algorithm’s run time is calculated in terms of the total number of steps or instructions
• Computed in terms of problem size, \( n \), and processor count, \( p \)
Example: Parallel Sum

- *REW*: A tree structure, leading to $\log n$ steps
- CRCW (associative): Single step (assuming reduction operator to be addition)
Definitions

- Run time of the parallel algorithm, $T_p$
- Run time of the best known sequential algorithm for the same problem, $T_s$ or $T_1$
- Cost, $C$: product of processor count and parallel execution time, $p \times T_p$
- Speed up, $S$: ratio of sequential to parallel running time, $T_1 / T_p$
- Efficiency, $E$: ratio of speed up to number of processors, $S / p$
- A cost optimal algorithm is one whose cost is equal to the sequential running time, $C = T_1$
- Therefore, $S = p$ and $E = 1$
Cost optimality

• Sorting: $T_1 = O(n \log n)$

• Cost optimal parallel sorting algorithms
  • uses $O(n)$ processors for $O(\log n)$ steps each
  • uses $O(n / \log n)$ processors for $O(\log^2 n)$ time

• An algorithm using $O(n^2)$ processors for $O(1)$ time runs faster but is not cost optimal

• Cost optimal algorithms scale better to machines with fewer processors
Speed up

- $S = \frac{T_1}{T_p}$, cannot be more than $p$
- Linear/ideal speed up: $S = p$
- An algorithm with linear speed up will run $k$ times as fast on $k$ times as many processors
- Cost optimal algorithms have linear speedup
Amdahl's law

- Assume an algorithm with sequential running time of 1, i.e. $T_1 = 1$
- Assume that a fraction, $P$, of the algorithm can be parallelized using $p$ processors
- Overall execution time = time for sequential portion + time for parallel portion = $(1 - P) + P/p$
- Therefore, overall speedup is $1 / ( (1 - P) + P/p )$
Estimating $P$

- For observed speed up, $S’$ on $p’$ processors, the estimated value of $P$ is 
  \[
  \frac{1}{S’} - 1 \bigg) / \left( \frac{1}{p’} - 1 \right) 
  \]

- Amdahl’s law assumes fixed problem size. $P$ stays fixed irrespective of $p$
Diminishing gains

- Image from wikipedia page on Amdahl's law
Gustafson’s law

- Relaxes Amdahl’s law by allowing problem size to vary
- Assume an algorithm with a serial portion, $S(n)$, where $n$ is the problem size
- Assume a total parallel running time of 1. So the parallelized portion takes time $1 - S(n)$
- Assuming $p$ processors, the sequential running time is $S(n) + p(1 - S(n))$
- Speedup is $S(n) + p(1 - S(n))$
- Assuming $S(n)$ diminishing with $n$, speed up approaches $p$
Brent’s theorem

• Assume an algorithm taking $t$ time steps to perform $m$ operations

• Let $s_i$ be the number of operations performed in time step $i$

  • $\sum_{i=1}^{t} s_i = m$

• With $p$ processors, $s_i$ can be done in time $\left\lceil \frac{s_i}{p} \right\rceil$

  • $\sum_{i=1}^{t} \left\lceil \frac{s_i}{p} \right\rceil \leq \sum_{i=1}^{t} (\frac{s_i}{p} + p - 1) / p = t + (m - t)/p$
Project Presentations

- 5 to 10 minutes each
  - Alexander Voigt - Compression
  - Thomas Rumpf - Game of Life
  - Philipp Lucas - EM Wave Propagation
  - T. Beier, M. Kaiser, L. Kuehne - Video Editing
  - Thomas Prinz - Virtual Machine
See you next time!