CSE 613: Parallel Programming

Lecture 19
(Optimizing Energy Consumption)

Rezaul A. Chowdhury
Department of Computer Science
SUNY Stony Brook
Spring 2012

Energy Consumption of Parallel Algorithms

We will try to analyze energy consumption of parallel algorithms on shared memory multicore processors.

In particular, we will look at a methodology to evaluate how energy consumption of a given parallel algorithm changes as the number of cores and their frequency is varied.

This lecture is based on the results presented in the following paper:

Vijay Anand Korthikanti and Gul Agha, "Towards optimizing energy costs of algorithms for shared memory architectures", Proceedings of the 22nd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), pp. 157-165, 2010.

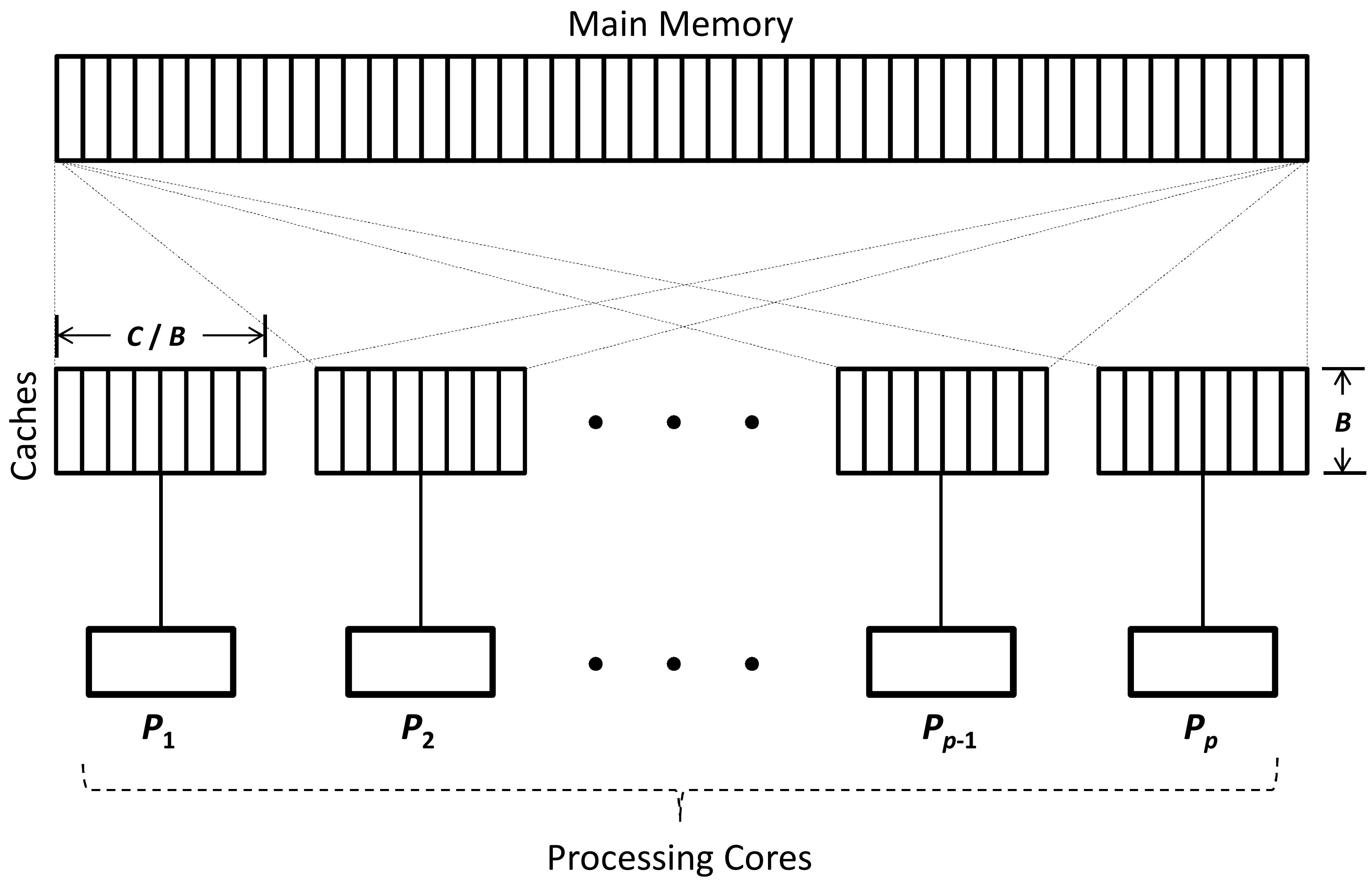
Energy Scalability under Iso-Performance

Question: *Given a problem instance and a fixed performance requirement, what is the number of cores that minimizes energy consumption in executing a parallel algorithm on the problem instance?*

A *problem instance* is a problem for a fixed input value.

The *performance* of a parallel algorithm on a problem instance is the time required for the completion of that problem instance.

Parallel Computation Model



Parallel External Memory (PEM) Model

Parallel Computation Model: Assumptions

— shared memory allows CREW (concurrent read, exclusive write)

— all active cores operate at the same frequency

— core frequencies can be varied using a frequency (voltage) probe

— cores switch to idle state if no computation left at them

— computation and cache access time of cores can be scaled by scaling core frequencies

— shared memory access (read & write) cannot be scaled and thus each takes constant time

Energy Model

Power consumption in a CMOS circuit (approximation):

$$\begin{tikzcd} & P = P_{dynamic} + P_{leakage} \arrow[dl] \arrow[dr] & \\ P_{dynamic} = C_L V^2 f & & P_{leakage} = I_L V \end{tikzcd}$$

where C_L = load capacitance,

V = supply voltage,

I_L = leakage current,

and f = operational frequency.

Energy Model

Let X be the frequency of a core. Then

computation time, $T_{busy} = \frac{\#(computation\ cycles)}{X}$

Let T_{active} = time for which a given core is active (not idle)

Then energy consumed by a core:

dynamic energy, $E_{dynamic} = E_d \cdot T_{busy} \cdot X^3$

leakage energy, $E_{leakage} = E_l \cdot T_{active} \cdot X$

where E_d and E_l are some hardware constants.

Energy Model: Assumptions

— idle cores do not consume energy

— a shared memory access (read & write) consume a constant amount of energy

Methodology

Step 1: Find the critical path π_A in the execution of A.

Step 2: Partition π_A into

(1) memory accesses,
(2) synchronization breaks, and
(3) computation steps.

Step 3: Scale the computation steps of π_A so that the parallel performance of A matches the specified performance requirement.

We scale the computation time of A to the difference between:

(*a*) the required performance, and
(*b*) time for memory access and synchronization breaks in π_A

Thus we get the reduced frequency X at which all cores should run.

Methodology

Step 4: Find the total number of computation cycles at all cores.

Step 5: Find the total number of memory accesses of A.

Step 6: Find the total active time of all cores at frequency X obtained in step 3.

Step 7: Find expressions for energy consumption:

$$E_{comp} = E_d \cdot \#(computation\ cycles) \cdot X^2$$
$$E_{mem} = E_m \cdot \#(memory\ acceses)$$
$$E_{leakage} = E_l \cdot T_{active} \cdot X$$

where E_m is the energy consumed for a single memory access.

Step 8: Analyze the equations above to obtain #cores required for minimum energy consumption as a function of input size.

Example: Adding *N* Numbers

Initial State: N number occupy contiguous locations in shared main memory, and all caches are empty.

Phase 1: Each core transfers $\frac{N}{p}$ numbers from main memory to its cache and sums them up in a series of steps.

In each step a core

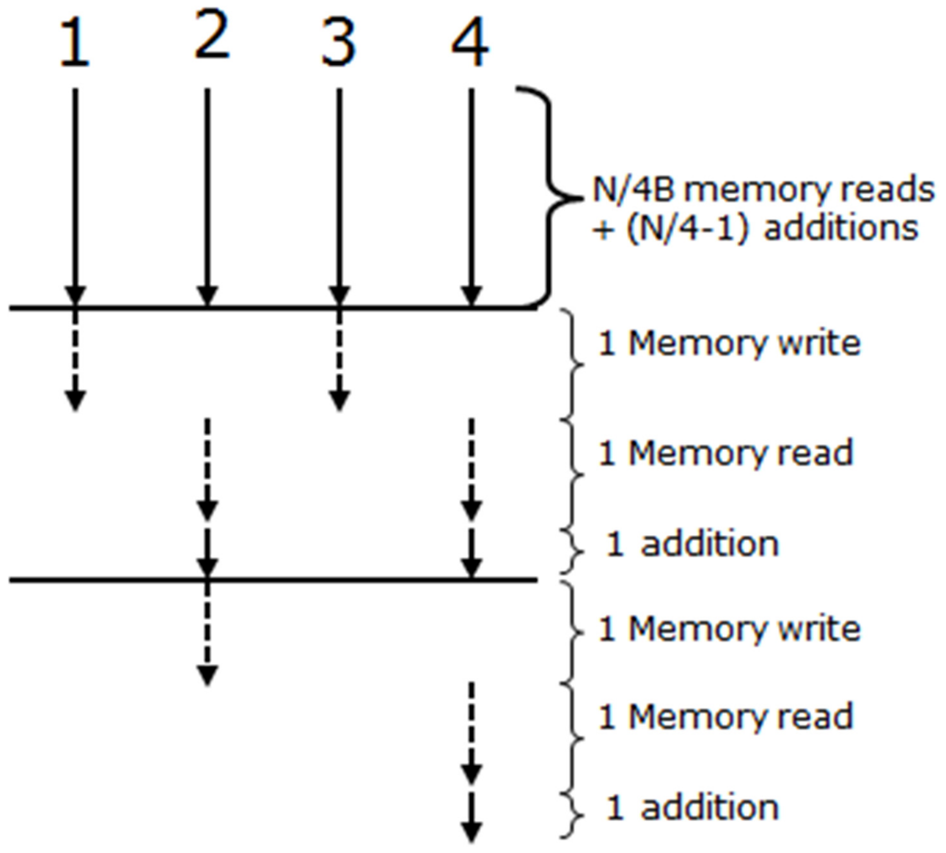
(1) transfers a block of B numbers from main memory, and
(2) computes the sum of those B numbers and the result obtained in the previous step

Phase 2: The sum of p partial sums from phase 1 is computed in parallel in a tree like fashion in $\log p$ steps.

In step $i \in [0, \log p]$, only $\frac{p}{2^i}$ cores are active

(1) half of them write their partial sums to main memory, and
(2) the other half read them, and add them to their own sum

Example: Adding *N* Numbers using 4 Cores



Example: Adding *N* Numbers

Step 1: Critical path π_A is the execution path of the core holding the final sum.

Step 2: We partition π_A, and find that there are

(1) $\frac{N}{pB} + \log p$ memory accesses,
(2) $\log p$ synchronization breaks, and
(3) $\frac{N}{p} - 1 + \log p$ computation steps.

Step 3: The reduced frequency: $X = F \cdot \frac{\left(\frac{N}{p} - 1 + \log p\right) \cdot \beta}{T \cdot F - \left(\frac{N}{pB} + 2 \log p\right) \cdot M_c}$

where T = required execution time,

F = maximum frequency of a single core,

β = number of cycles per addition, and

M_c = #cycles at frequency F for each shared memory access

Example: Adding *N* Numbers

Step 4: #comp. cycles = $\left(\left(\frac{N}{p}-1\right)\cdot p+(p-1)\right)\cdot\beta=(N-1)\cdot\beta$

Step 5: #memory accesses = $\frac{N}{B}+2(p-1)$

Step 6: $T_{active}=\frac{M_c}{F}\left(\frac{N}{B}+2(p-1)\right)+\frac{\beta}{X}\cdot(N-1)$

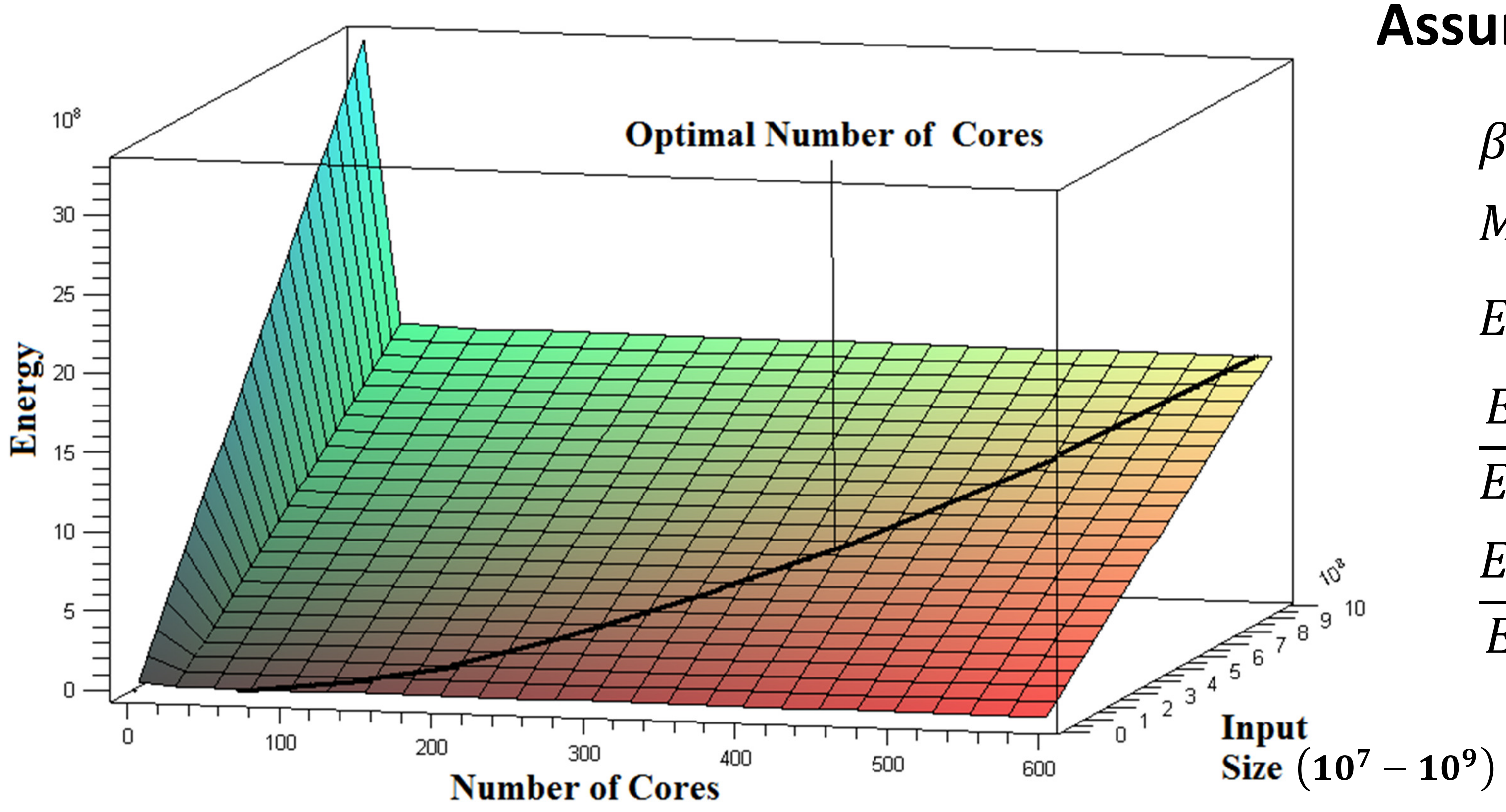
Step 7: Expressions for energy consumption:

$$E_{comp}=E_d\cdot(N-1)\cdot\beta\cdot X^2$$

$$E_{mem}=E_m\cdot\left(\frac{N}{B}+2(p-1)\right)$$

$$E_{leakage}=E_l\cdot\left(\frac{M_c}{F}\left(\frac{N}{B}+2(p-1)\right)+\frac{\beta}{X}\cdot(N-1)\right)\cdot X$$

Example: Adding *N* Numbers



Assumptions:

$$\beta = 2$$

$$M_c = 1000$$

$$E_l = 1$$

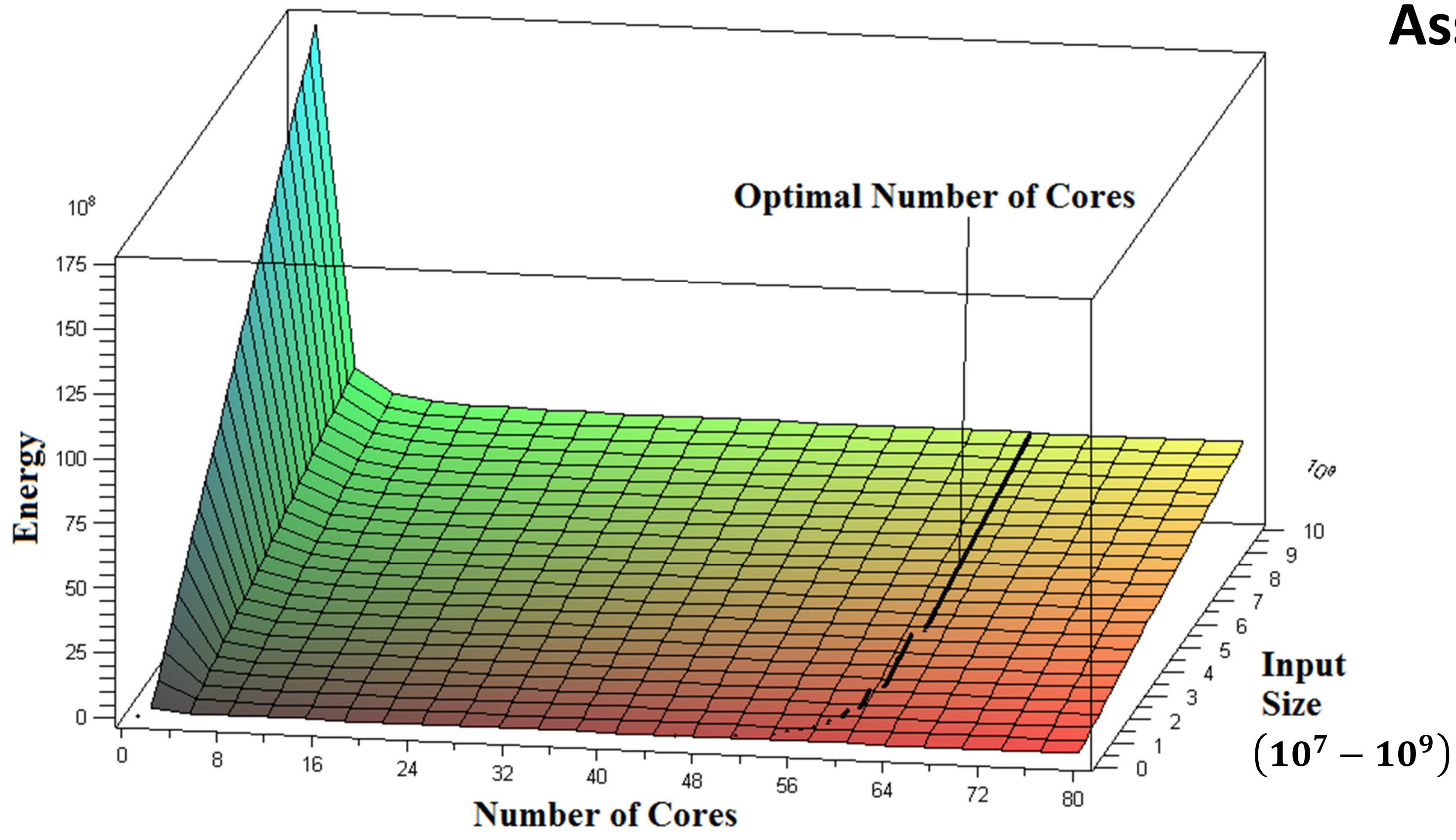
$$\frac{E_l}{E_d} = \frac{1}{10} \cdot F^2$$

$$\frac{E_m}{E_d} = 1000 \cdot F^2$$

Performance Requirement: Running time of the sequential algorithm at maximum frequency F:

$$T = T_{sequential} = \frac{\beta}{F} \cdot (N - 1) + \frac{M_c}{F} \cdot \frac{N}{B}$$

Example: Parallel Prefix Sums



Assumptions:

$$\beta = 2$$

$$M_c = 1000$$

$$E_l = 1$$

$$\frac{E_l}{E_d} = \frac{1}{10} \cdot F^2$$

$$\frac{E_m}{E_d} = 1000 \cdot F^2$$

Performance Requirement: Running time of the sequential algorithm at maximum frequency F:

$$T = T_{sequential} = \frac{\beta}{F} \cdot (N - 1) + \frac{M_c}{F} \cdot \frac{N}{B}$$

Example: Parallel Merge Sort

For the *pipelined d-way mergesort algorithm* (developed by Arge et al., 2008, for the PEM model) considered in the paper:

— E_{comp} decreases as p increases, and

— $E_{leakage}$ also decreases as p increases, but

— E_{mem} is independent of p

So, energy consumed by the parallel algorithm to maintain the same performance as the sequential algorithm decreases with increasing p (under the assumption that $p \leq \frac{N}{B^2}$).

This observation can be generalized to a general class of parallel algorithms with optimal computation cost and optimal I/O complexity.