CS 598: Provably Efficient Algorithms for Numerical and Combinatorial Problems

Part 4: Communication Cost in Algorithms

Edgar Solomonik

University of Illinois at Urbana-Champaign

Algorithmic cache management

Consider a computer with unlimited memory and a cache of size H

- we can design algorithms by manually managing cache transfers
- generally, efficient algorithms in this model try to select blocks of computation that minimize the surface-to-volume ratio

Cache-efficient matrix multiplication

Consider multiplication of $n \times n$ matrices $\boldsymbol{C} = \boldsymbol{A} \cdot \boldsymbol{B}$

For $i \in [1, n/s], j \in [1, n/t]$, $k \in [1, n/v]$, define blocks $\boldsymbol{C}[i, j]$, $\boldsymbol{A}[i, k]$, $\boldsymbol{B}[k, j]$ with dimensions $s \times t$, $s \times v$, and $v \times t$, respectively

```
for (i = 1 to n/s)
  for (j = 1 to n/t)
    initialize C[i,j] = 0 in cache
    for (k = 1 to n/v)
      load A[i,k] into cache
      load B[k,j] into cache
      C[i,j] = C[i,j] + A[i,k]*B[k,j]
    end
    write C[i,j] to memory
  end
end
```

Memory-bandwidth analysis of matrix multiplication

- Lets consider bandwidth and latency cost if each matrix multiplication has dimensions s, t, v
- Given the constraint, $st + sv + vt \leq H$, we can derive the optimal block sizes

Ideal cache model

- A more accurate model is to consider a cache line size L in addition to the cache size H
- We can now consider different caching protocols

Matrix transposition in the ideal cache model

- Matrix multiplication bandwidth cost with a tall cache is not affected by L

- $n \times n$ matrix transposition becomes non-trivial

Cache obliviousness

- Introduced by Frigo, Leiserson, Prokop, Ramachadran

- cache oblivious algorithms are stated without explicit control of data movement

Cache oblivious matrix transposition

Given $m \times n$ matrix $\boldsymbol{A}$, compute $\boldsymbol{B} = \boldsymbol{A}^T$

Cache oblivious matrix multiplication

Given $m \times k$ matrix $\boldsymbol{A}$ and $k \times n$ matrix $\boldsymbol{B}$, compute $m \times n$ matrix $\boldsymbol{C} = \boldsymbol{AB}$

Cache oblivious fast Fourier transform (FFT)

- The Fourier transform computes $\boldsymbol{y} = \boldsymbol{D}^{(n)}\boldsymbol{x}$, where $d_{ij}^{(n)} = \omega_n^{ij}$ and ω_n is the nth complex root of identity
- A cache-oblivious algorithm for the FFT can be derived by folding $\boldsymbol{y}$ and $\boldsymbol{x}$ into matrices $\boldsymbol{Y}$ and $\boldsymbol{X}$ of dimensions $\sqrt{n} \times \sqrt{n}$

Cache oblivious fast Fourier transform (FFT)

- Lets now analyze the cost of the cache oblivious algorithm based on

$$\boldsymbol{Y} = (((\boldsymbol{D}^{(m)}\boldsymbol{X}) \odot \boldsymbol{F})\boldsymbol{D}^{(m)})^T$$

A simple model for point-to-point messages

The time to send or receive a message of s words is $\alpha + s \cdot \beta$ Consider the cost of

a broadcast of s words

Bulk Synchronous Parallel (BSP) Model

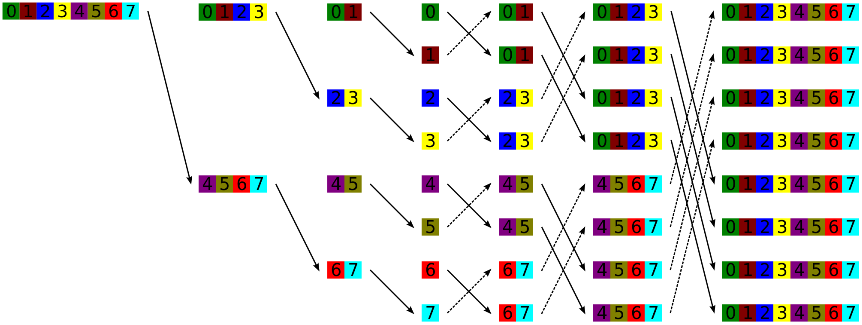
- *Bulk Synchronous Parallel (BSP) model* (Valiant 1990)

- The cost of a BSP algorithm is a sum over supersteps of the maximum costs incurred in that superstep

Collective communication in BSP

- When $h = p$, most collective communication routines involving s words of data per processor can be done with BSP cost $O(\alpha + s \cdot \beta)$

Butterfly Broadcast



Matrix-vector Product

- Lets design a cache-efficient algorithm for a matrix-vector product

- Lets design a BSP algorithm for a matrix-vector product

Sparse matrix-vector Product

- 1D distribution is effective for BSP algorithm for SpMV

Massively Parallel Computing (MPC) Model

- Massively Parallel Computing (MPC) model

- Aim to achieve $O(\log N)$ or $O(\log \log N)$ rounds with minimal memory per processor

Graph Algorithms in MPC

- Graph algorithms in the MPC model for graphs with n vertices

Communication lower bounds

- Given an algorithm (e.g. radix-2 FFT, bitonic sort) or family of algorithms (e.g. radix-k FFT, comparison based sorting algorithms), how much communication is necessary?
- Communication lower bounds ascertain optimality of communication schedules

Classical results in communication lower bounds

- Floyd 1972: for large cache lines $L = \Theta(H)$
- Hong and Kung 1981, pebbling lower bound
- Aggarwal and Vitter 1988, lower bounds with any L, H

Lower bounds by partitioning memory operations

Pebbling bounds employ the following general argument

Lower bounds by partitioning computation

We can also take the dual view

- we are given an algorithm that must perform F operations
- we need to prove that the given $3H$ inputs and outputs at most $f_{\mathsf{alg}}(H)$ of the computation can be done

Bounding work in matrix multiplication

Consider the $F = n^3$ products computed in square matrix multiplication

Cache complexity lower bound for MM

Given $f_{\mathsf{MM}}(H) = H^{3/2}$, we are essentially done

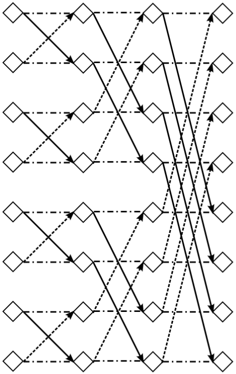
Interprocessor communication lower bound for MM

We can also use f_{MM} to get lower bounds on interprocessor communication

Latency/synchronization lower bounds

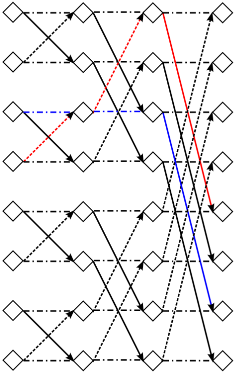
From f_{MM} to get lower bounds on the number of messages

Radix-2 FFT dependency graph



Paths in Radix-2 FFT dependency graph

Any two edge-disjoint paths in the FFT DAG intersect at no more than one vertex

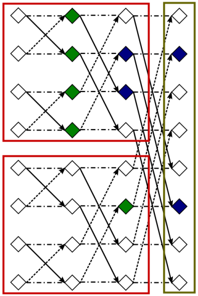


in other words, the FFT DAG has no cycles

Work bound for FFT

We prove that the work bound for the radix-2 FFT is $f_{\mathsf{FFT}}(s) = s \log_2 s$

Work bound for FFT, contd



Communication lower bound for the FFT

By induction the expression $f_{\mathsf{FFT}}(s) = \max_t(f_{\mathsf{FFT}}(s-t) + f_{\mathsf{FFT}}(t) + 2\min(s-t,t))$ implies

$$f_{\mathsf{FFT}}(s) = \max_t((s-t)\log_2(s-t) + t\log(t) + 2\min(s-t,t))$$

Lower bounds via graph partitioning

- Given a DAG representation of an algorithm, graph partitioning properties can provide communication lower bounds
- Consideration of expansion of subgraphs can yield better bounds

Dependency interval expansion

Consider an algorithm that computes a set of operations V with a partial ordering, we denote a dependency interval between $a, b \in V$ as

$$[a, b] = \{a, b\} \cup \{c : a < c < b, c \in V\}$$

Dependency interval expansion

Consider an algorithm that computes a set of operations V with a partial ordering, we denote a dependency interval between $a, b \in V$ as

$$[a, b] = \{a, b\} \cup \{c : a < c < b, c \in V\}$$

Dependency interval expansion

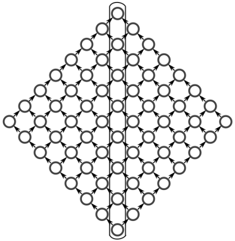
Consider an algorithm that computes a set of operations V with a partial ordering, we denote a dependency interval between $a, b \in V$ as

$$[a, b] = \{a, b\} \cup \{c : a < c < b, c \in V\}$$

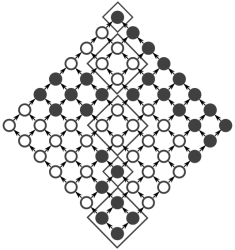
Further, if the algorithm has a work bound $f(H) = \Omega(H^{\frac{d}{d-1}})$, then

$$W \cdot S^{d-2} = \Omega(n^{d-1})$$

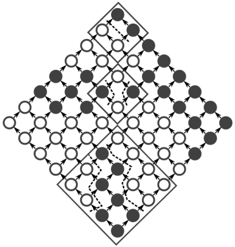
Example: diamond DAG



Dependency chain P



Monochrome dependency intervals



Multicolored dependency intervals

For the $n \times n$ diamond DAG ($d = 2$),

$$
\begin{aligned}
F \cdot S^{d-1} &= F \cdot S = \Omega((n/b)b^2) \cdot \Omega(n/b) = \Omega(n^2) \\
W \cdot S^{d-2} &= W \quad = \Omega((n/b)b) \qquad\qquad = \Omega(n)
\end{aligned}
$$

idea of $F \cdot S$ tradeoff goes back to Papadimitriou and Ullman, 1987

Tradeoffs involving synchronization

For triangular solve with an $n \times n$ matrix For Cholesky of an $n \times n$ matrix For

computing s applications of a $(2m+1)^d$-point stencil