CS 598: Provably Efficient Algorithms for Numerical and Combinatorial Problems

Part 2: Algorithm Representation

Edgar Solomonik

University of Illinois at Urbana-Champaign

Straight Line Programs

- Often, we want to quantify the efficiency of an algorithm that solves any problem of size n in $f(n)$ iterations, i.e., it is a *straight line program*
  - *Numerical algorithms often fit this form, including for matrix multiplication, convolution, matrix factorizations, direct solvers, k-iterations of an sparse iterative method*
  - *Programs that have branches or conditional loop bounds may not be described a straight line program*
- The completed execution of a program for a particular problem may always be described by a straight line program
  - *Ultimately a sequence of instructions are executed*
  - *These instructions may be independent and so can be reordered, thus we have a set of operations which need to be executed according to a partial order*

Algorithms as Directed Acyclic Graphs

- A directed acyclic graph (DAG) describes a straight line program in terms of elementwise operations (addition, multiplication, etc.)
  - *This DAG has a vertex for each scalar value input to or computed within the program*
  - *Computed values are vertices with in-degree one or two in the DAG*
- Assuming an algorithm is a straight line program, we may ask questions regarding parallelism and communication cost
  - *Depth of DAG gives lower bound on parallel execution time*
  - *Expansion properties describe communication cost*

Schedules of an Algorithm

- A *schedule* assigns the vertices of a straight-line program to instructional units and maanages associated communication
  - *Schedule depends on architectural model*
    - *Sequential with explicitly managed bounded fast memory (cache/register file), schedule needs to provide reads/writes/discards*
    - *Sequential with implicitly managed bounded fast memory (cache/register file) via a given caching protocol*
    - *Parallel shared memory with no fast memory (PRAM), variants regarding how to handle concurrent reads/writes to same locations*
    - *Distributed-memory parallel (BSP/α–β), schedule may manage initial data layout, communication and synchronization*

Parameterization of Algorithms

- Oftentimes, we may want to paramterize the algorithm (and not just the schedule) depending on the architecture
  - *For example, we may use a flat reduction tree on a single processor as opposed to a binary reduction tree with many processors*
  - *Reorganization of DAG may minimize expansion with respect to a parameter controlling subset size (which may be correlated with fast memory size)*
- An algorithm may also be designed to be *oblivious* to a parameter, i.e., to minimize execution time for any choice of a particular parameter
  - *This notion is most important for fast memory (cache) size, with corresponding algorithms refered to as cache-oblivious*
  - *Other examples include network-oblivious algorithms*

Matrix Multiplication as a DAG

Lets consider the matrix multiplication problem: compute $\boldsymbol{C}$ such that $\boldsymbol{C} = \boldsymbol{AB}$ with $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C} \in \mathbb{R}^{n\times n}$

- Loop-nest can be used to describe algorithm/DAG (for i, for j, for k, $c_{ij}^{(k)} = c_{ij}^{(k-1)} + a_{ik}b_{kj}$ with $c_{ij}^{(0)} = 0$ and $c_{ij} = c_{ij}^{(n)}$)
  - *As stated, loop nest describes flat reduction tree with depth* $O(n)$
  - *Using associativity of addition can obtain DAGs with depth* $O(\log(n))$
- Recursive formulation describes another algorithm/DAG

$$\begin{bmatrix} \boldsymbol{C}_{11} & \boldsymbol{C}_{12} \\ \boldsymbol{C}_{21} & \boldsymbol{C}_{22} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} \\ \boldsymbol{A}_{21} & \boldsymbol{A}_{22} \end{bmatrix} \begin{bmatrix} \boldsymbol{B}_{11} & \boldsymbol{B}_{12} \\ \boldsymbol{B}_{21} & \boldsymbol{B}_{22} \end{bmatrix}$$

  - *Performing eight block products recursively yields cost*

$$T(n) = 8T(n/2) + O(n^2) = O(n^3)$$

  - *Recursive calls can be done in parallel and algorithm is cache-oblvious*

Family of Classical Matrix Multiplication Algorithms

- The nested-loop and recursive formulations are two instances of a family of classical matrix multiplication algorithms
  - *They yield different partial orders*
  - *Both compute products* $a_{ik}b_{kj}$ *and sum along* k
  - *Conseuqently, associativity of addition implies they are equivalent in exact arithmetic*
- Can describe family of DAGs as a hypergraph
  - *Define hyperedges* $h_{ij}^{(C)} = (\{a_{ik}b_{kj} : k \in \{1, \ldots, n\}\}, c_{ij})$ *to denote reduction/sum*
  - *Could further abstract DAG edges* $(a_{ik}, a_{ik}b_{kj})$ *as 'broadcast' hyperedges* $h_{ik}^{(A)} = (a_{ik}, \{a_{ik}b_{kj} : k \in \{1, \ldots, n\}\})$ *and* $h_{kj}^{(B)}$ *similarly*
  - *Expansion properties of this hypergraph imply expansion properties for DAGs arising from any summation order*

Surface Area to Volume Ratio in Hypergraphs

- We can analyze the hypergraph to determine communication cost bounds
  - *Would like to consider any partial order equivalent via associativity*
  - *For simplicity, often want to assume no intermediate values are recomputed*
- The *Loomis-Whitney* is a *volumetric inequality* that provides a way to bound expansion
  - *Consider a set of products* $S \subseteq \{a_{ik}b_{kj} : i, j, k \in \{1, \ldots, n\}\}$
  - *Let* $H_S = H_S^{(A)} \cup H_S^{(B)} \cup H_S^{(C)}$ *be the set of hyperedges adjacent to* S*,*

$$|S| \leq \left(|H_S^{(A)}| \cdot |H_S^{(B)}| \cdot |H_S^{(C)}|\right)^{1/2}$$

  - *Consequently, we can derive communication lower bounds by infering that e.g.,*

$$|H_S| \geq (1/3)^{1/3}|S|^{2/3}$$

Compression and Recomputation

- Our previous discussion of communication assumed that each hypergraph edge requires communication of a matrix entry
  - *Perhaps it is possible to communicate less information by transforming a set of entries, e.g. taking linear combinations there of (sending $a_{ij} + a_{i'j'}$ instead of both individually)?*
  - *Could lower bound information low and compression via linear combinations by considering rank*
- A method that computes bilinear products $a_{ik}b_{kj}$ may take arbitrary linear combinations of entries of $\boldsymbol{A}$, $\boldsymbol{B}$, or partial sums for $\boldsymbol{C}$
  - *Given a vector of input linear combinations $\boldsymbol{s}^{(A)}$ of entries of $\boldsymbol{A}$, $\boldsymbol{s}^{(B)}$ of entries of $\boldsymbol{B}$, output some set of linear combinations $\boldsymbol{s}^{(C)}$ of products $a_{ik}b_{kj}$, we have that for some $\boldsymbol{A}^{(S)}$, $\boldsymbol{B}^{(S)}$, $\boldsymbol{C}^{(S)}$,*

$$\boldsymbol{s}^{(C)} = \boldsymbol{C}^{(S)}\left[(\boldsymbol{A}^{(S)T}\boldsymbol{s}^{(A)}) \odot (\boldsymbol{B}^{(S)T}\boldsymbol{s}^{(B)})\right]$$

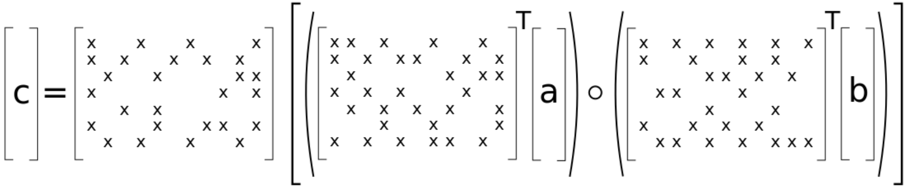
  - *To lower bound the dimensions of $\boldsymbol{s}^{(A)}$, $\boldsymbol{s}^{(B)}$, and $\boldsymbol{s}^{(C)}$, need to relate $\boldsymbol{A}^{(S)}$, $\boldsymbol{B}^{(S)}$, and $\boldsymbol{C}^{(S)}$ to overall computation*

Bilinear Algorithms

A bilinear algorithm (V. Pan, 1984) $\Lambda = (\boldsymbol{F}^{(A)}, \boldsymbol{F}^{(B)}, \boldsymbol{F}^{(C)})$ computes

$$\boldsymbol{c} = \boldsymbol{F}^{(C)}[(\boldsymbol{F}^{(A)T}\boldsymbol{a}) \odot (\boldsymbol{F}^{(B)T}\boldsymbol{b})],$$

where $\boldsymbol{a}$ and $\boldsymbol{b}$ are inputs and $\odot$ is the Hadamard (pointwise) product.



Bilinear Algorithms as Tensor Factorizations

- A bilinear algorithm corresponds to a CP tensor decomposition

$$c_i = \sum_{r=1}^{R} f^{(C)}_{ir} \left(\sum_j f^{(A)}_{jr} a_j \right) \left(\sum_k f^{(B)}_{kr} b_k \right)$$

$$= \sum_j \sum_k \left(\sum_{r=1}^{R} f^{(C)}_{ir} f^{(A)}_{jr} f^{(B)}_{kr} \right) a_j b_k$$

$$= \sum_j \sum_k t_{ijk} a_j b_k \quad \textit{where} \quad t_{ijk} = \sum_{r=1}^{R} f^{(C)}_{ir} f^{(A)}_{jr} f^{(B)}_{kr}$$

- For multiplication of $n \times n$ matrices, we can define a *matrix multiplication tensor* and consider algorithms with various bilinear rank
  - $\boldsymbol{T}$ *is* $n^2 \times n^2 \times n^2$
  - *Classical algorithm has rank* $R = n^3$
  - *Strassen's algorithm has rank* $R \approx n^{\log_2(7)}$

Strassen's Algorithm

Strassen's algorithm $\begin{bmatrix} \boldsymbol{C}_{11} & \boldsymbol{C}_{12} \\ \boldsymbol{C}_{21} & \boldsymbol{C}_{22} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} \\ \boldsymbol{A}_{21} & \boldsymbol{A}_{22} \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{B}_{11} & \boldsymbol{B}_{12} \\ \boldsymbol{B}_{21} & \boldsymbol{B}_{22} \end{bmatrix}$

$$
\begin{aligned}
\boldsymbol{M}_1 &= (\boldsymbol{A}_{11} + \boldsymbol{A}_{22}) \cdot (\boldsymbol{B}_{11} + \boldsymbol{B}_{22}) \\
\boldsymbol{M}_2 &= (\boldsymbol{A}_{21} + \boldsymbol{A}_{22}) \cdot \boldsymbol{B}_{11} \\
\boldsymbol{M}_3 &= \boldsymbol{A}_{11} \cdot (\boldsymbol{B}_{12} - \boldsymbol{B}_{22}) \\
\boldsymbol{M}_4 &= \boldsymbol{A}_{22} \cdot (\boldsymbol{B}_{21} - \boldsymbol{B}_{11}) \\
\boldsymbol{M}_5 &= (\boldsymbol{A}_{11} + \boldsymbol{A}_{12}) \cdot \boldsymbol{B}_{22} \\
\boldsymbol{M}_6 &= (\boldsymbol{A}_{21} - \boldsymbol{A}_{11}) \cdot (\boldsymbol{B}_{11} + \boldsymbol{B}_{12}) \\
\boldsymbol{M}_7 &= (\boldsymbol{A}_{12} - \boldsymbol{A}_{22}) \cdot (\boldsymbol{B}_{21} + \boldsymbol{B}_{22})
\end{aligned}
$$

$$
\begin{aligned}
\boldsymbol{C}_{11} &= \boldsymbol{M}_1 + \boldsymbol{M}_4 - \boldsymbol{M}_5 + \boldsymbol{M}_7 \\
\boldsymbol{C}_{21} &= \boldsymbol{M}_2 + \boldsymbol{M}_4 \\
\boldsymbol{C}_{12} &= \boldsymbol{M}_3 + \boldsymbol{M}_5 \\
\boldsymbol{C}_{22} &= \boldsymbol{M}_1 - \boldsymbol{M}_2 + \boldsymbol{M}_3 + \boldsymbol{M}_6
\end{aligned}
$$

By performing the nested calls recursively, Strassen's algorithm achieves cost,

$$T(n) = 7T(n/2) + O(n^2) = O(7^{\log_2 n}) = O(n^{\log_2 7})$$

Expansion in Bilinear Algorithms

- The communication cost of a bilinear algorithm depends on the amount of data needed to compute subsets of the bilinear products.
  - *A schedule may involve computations of parts of the bilinear algorithm, of the form,* $\Lambda = (\boldsymbol{F}^{(A)}, \boldsymbol{F}^{(B)}, \boldsymbol{F}^{(C)})$*,* $\Lambda_{\text{sub}} \subseteq \Lambda$ *if for some projection matrix* $\boldsymbol{P}$*,*

$$\Lambda_{\text{sub}} = (\boldsymbol{F}^{(A)}\boldsymbol{P}, \boldsymbol{F}^{(B)}\boldsymbol{P}, \boldsymbol{F}^{(C)}\boldsymbol{P}).$$

  - *The projection matrix extracts* $\#\text{cols}(\boldsymbol{P})$ *columns of each matrix.*
- A bilinear algorithm Λ can be associated expansion bound $\mathcal{E}_\Lambda : \mathbb{N}^3 \to \mathbb{N}$
  - *Expansion bounds holds if for all*

$$\Lambda_{\text{sub}} := (\boldsymbol{F}^{(A)}_{\text{sub}}, \boldsymbol{F}^{(B)}_{\text{sub}}, \boldsymbol{F}^{(C)}_{\text{sub}}) \subseteq \Lambda$$

    we have $\text{rank}(\Lambda_{\text{sub}}) \leq \mathcal{E}_\Lambda\left(\text{rank}(\boldsymbol{F}^{(\boldsymbol{A})}_{\textbf{sub}}), \text{rank}(\boldsymbol{F}^{(\boldsymbol{B})}_{\textbf{sub}}), \text{rank}(\boldsymbol{F}^{(\boldsymbol{C})}_{\textbf{sub}})\right)$
  - *For matrix mult., Loomis-Whitney inequality* $\to \mathcal{E}_{\mathit{MM}}(x, y, z) = \sqrt{xyz}$