A Many-core Machine Model for Designing Algorithms with Minimum Parallelism Overheads

Sardar Anisul Haque, Marc Moreno Maza and Ning Xie

University of Western Ontario, Canada

HPCS 2013 – University of Ottawa
June 6, 2013

Parallelism Overheads and Models of Computations

Why is my parallel program not reaching linear speedup? not scaling?

- The algorithm could lack of parallelism ...
- The architecture could suffer from limitations ...
- The program does not expose enough parallelism ...
- Or the concurrency platform suffers from overheads (such as communication and synchronization costs)!

Challenges for models of computations

- Retaining the features of actual computers that have a dominant impact on program performance is hard
- Using several complexity measures (work, span, cache complexity) is necessary, but
- how to combine those complexity measures in order to select the "best" algorithm among several candidates for a given problem?
- Parallelism overheads are often ignored or included by other performance counters.

Models of Computations Targeting Many-core Architectures

Popular models

- PRAM (parallel random access machine) supports data parallelism but not task parallelism. Moreover, cannot support memory traffic issues (cache complexity, memory contention)
- Queue Read Queue Write PRAM considers memory contention, however, it unifies in a single quantity time spent in arithmetic operations and time spent in read/write accesses
- TMM (Threaded Many-core Memory) model retains many important characteristics of GPU-type architectures, however, the running time estimate on P cores is not given by a Graham-Brent theorem

In this work:

We propose a many-core machine model (MMM) which aims at optimizing algorithms targeting implementation on GPUs. We insist on

- Two-level DAG (directed acyclic graph) programs
- Parallelism overhead
- A Graham-Brent theorem

Motivations and Strategy

Motivations

- At HPCS 2012, we reported on an optimized GPU implementation of polynomial arithmetic operations (division, GCD, multiplication)
- These optimizations were obtained by minimizing data transfer between global and local memories and also by minimizing the impact of code divergence in kernels

Using the MMM

For algorithm analysis,

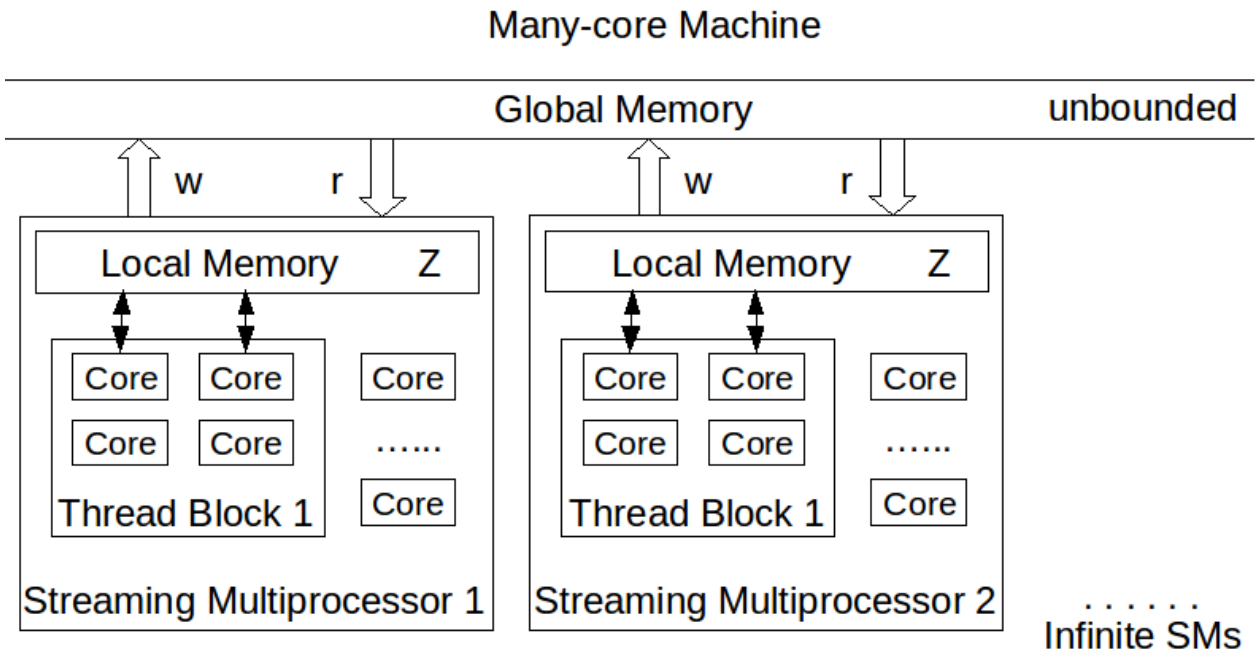
1) we determine a value of a program parameter that minimizes parallelism overhead
2) we check that the work overhead introduced by this optimization technique remains low
3) we use our version of Graham-Brent theorem to show that the estimated running time on p cores of the optimized algorithm is asymptotically smaller than that of the non-optimized algorithm

Contents

Many-core Machine Model

- Characteristics
- Programs
- Complexity Measures
- Graham-Brent Theorem with Overhead

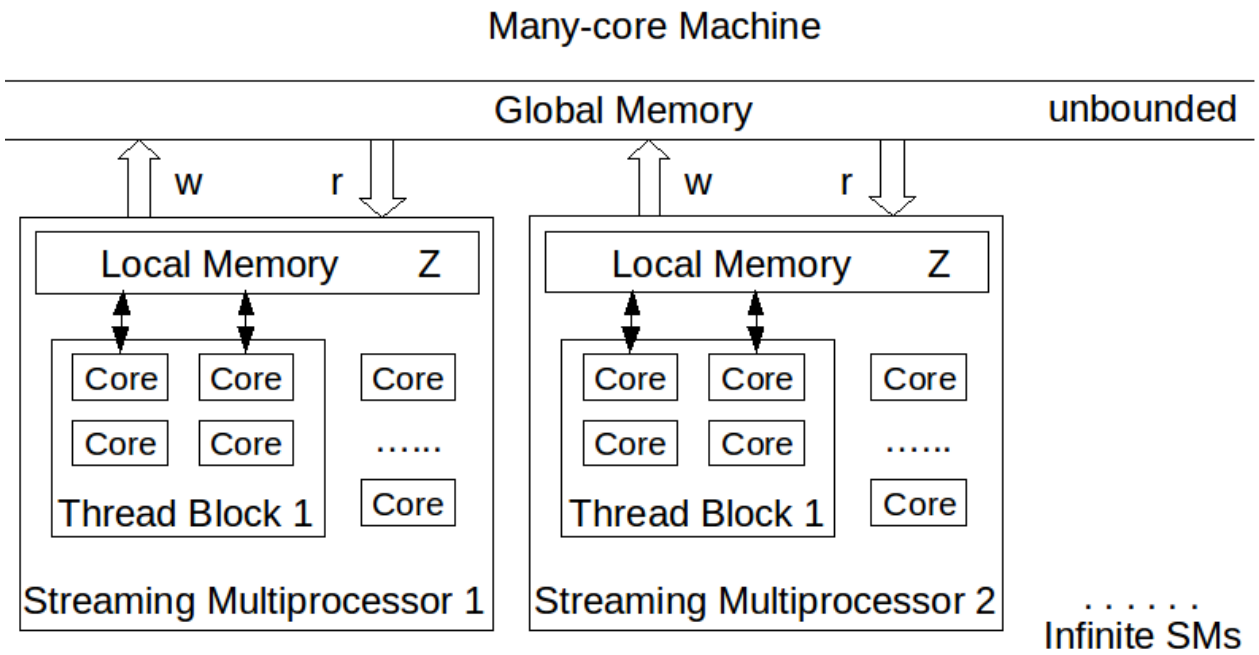
MMM: Characteristics



Architecture:

- Unbounded number of *streaming multiprocessors* (SMs) which are all identical
- Each SM has a finite number of processing cores and a fixed-size local memory
- 2-level memory hierarchy, comprising an unbounded global memory with high latency and low throughput while the SM local memories have low latency and high throughput

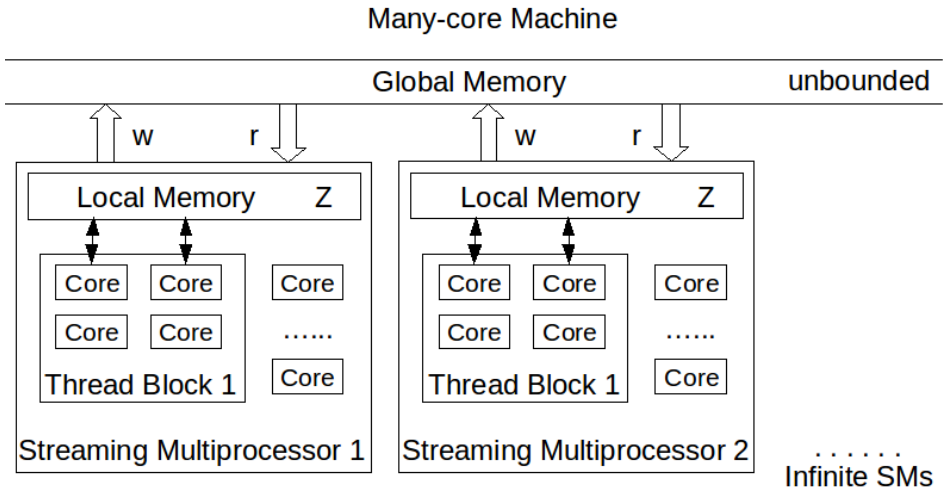
MMM: Characteristics



Programs:

- An MMM program is a directed acyclic graph (DAG) whose vertices are kernels and where edges indicate dependencies
- A kernel is a SIMD (single instruction multithreaded data) program decomposed into a number of thread-blocks
- Each thread-block is executed by a single SM and each SM executes a single thread-block at a time

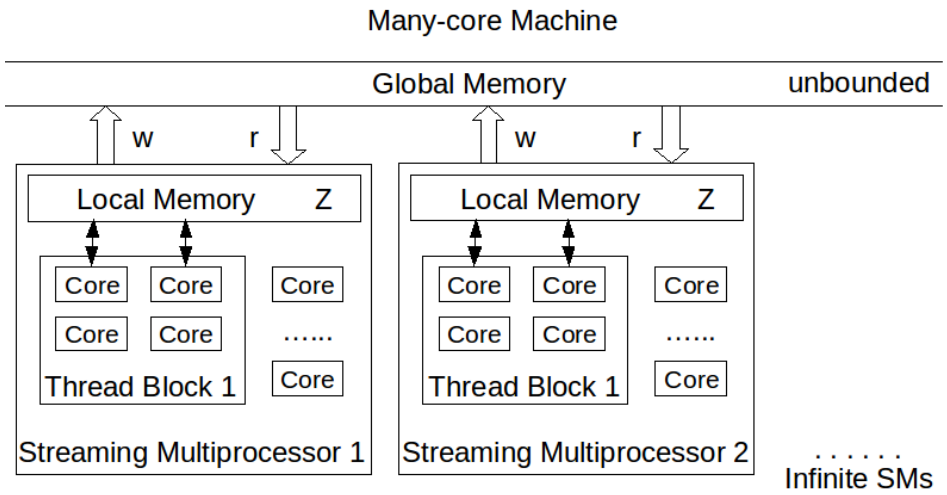
MMM: Characteristics



Scheduling and synchronization:

- At run time, an MMM machine schedules thread-blocks onto the SMs, based on the dependencies among kernels and the hardware resources required by each thread-block
- Threads within a thread-block cooperate with each other via the local memory of the SM running the thread-block
- Thread-blocks interact with each other via the global memory

MMM: Characteristics



Memory access policy:

- All threads of a given thread-block can access simultaneously any memory cell of the local memory or the global memory
- Read/Write conflicts are handled by the CREW (concurrent read exclusive write) policy

MMM: Characteristics

For the purpose of analyzing program performance, we define two *machine parameters*

- U: Time (expressed in clock cycles) to transfer one machine word between global memory and the local memory of any SM
- Z: Size (expressed in machine words) of the local memory of each SM

For a thread-block B, if each thread executes at most ℓ local (i.e. arithmetic) operations, and reads r (resp. writes w) words to the global memory, then to compute the total running time T of an SM executing B,

- the total time T_D spent in data transfer between the global memory and the local memory

$$T_D \leq (r + w)\, U$$

- there exists a constant V such that the total time T_A spent in local operations satisfies

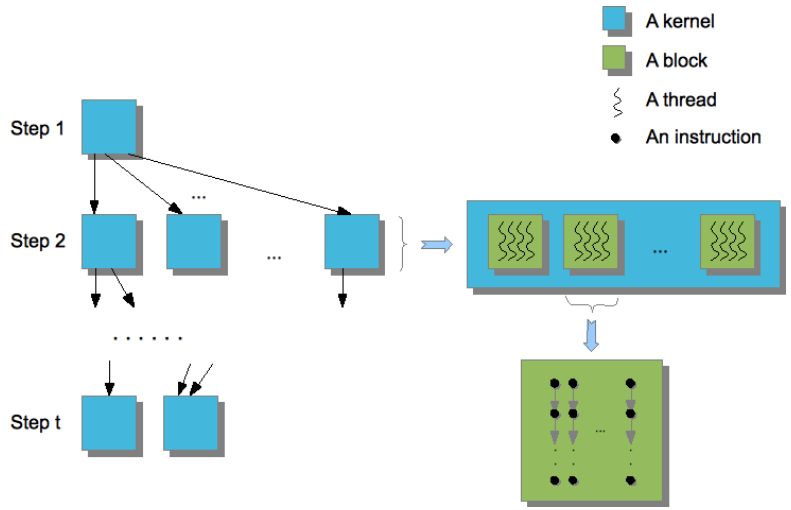
$$T_A \leq \ell\, V$$

we have

$$T = T_A + T_D \leq \ell + (r + w)\, U, \text{ with } V = 1.$$

MMM: Programs

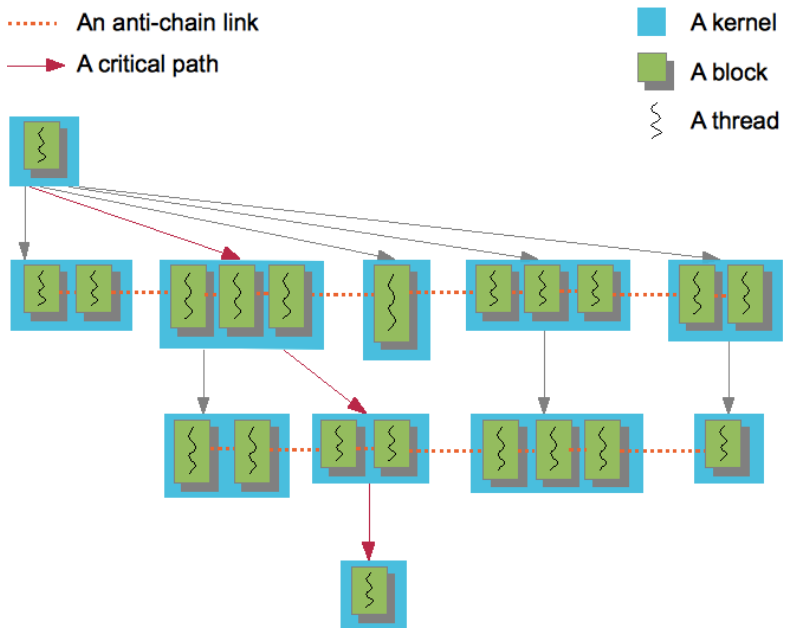
Each MMM program $\mathcal{P}$ is modeled by a directed acyclic graph $(\mathcal{K}, \mathcal{E})$, called the **kernel DAG** of $\mathcal{P}$, where each node $K \in \mathcal{K}$ represents a kernel, and each edge $E \in \mathcal{E}$ represents a kernel call which must precede another kernel call.



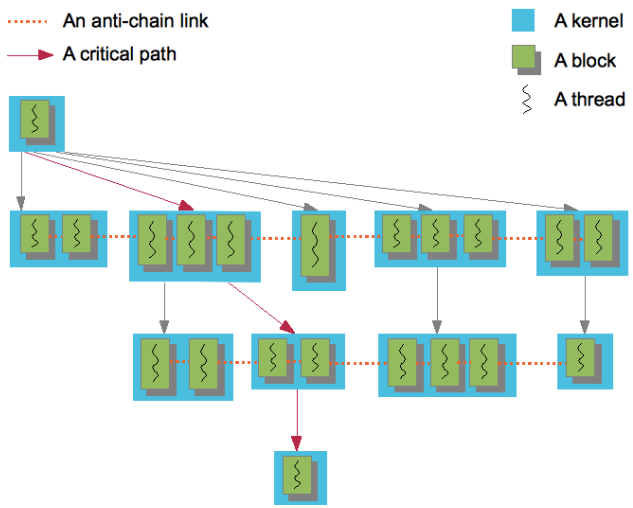
- Note: a kernel call can be executed whenever all its predecessors in the DAG $(\mathcal{K}, \mathcal{E})$ have completed their execution

MMM: Programs

Since each kernel of the program $\mathcal{P}$ decomposes into a finite number of thread-blocks, we map $\mathcal{P}$ to a second graph, called the **thread block DAG** of $\mathcal{P}$, whose vertex set $\mathcal{B}(\mathcal{P})$ consists of all thread-blocks of the kernels of $\mathcal{P}$, such that (B_1, B_2) is an edge if B_1 is a thread-block of a kernel preceding the kernel of B_2 in P.



MMM: Complexity Measures

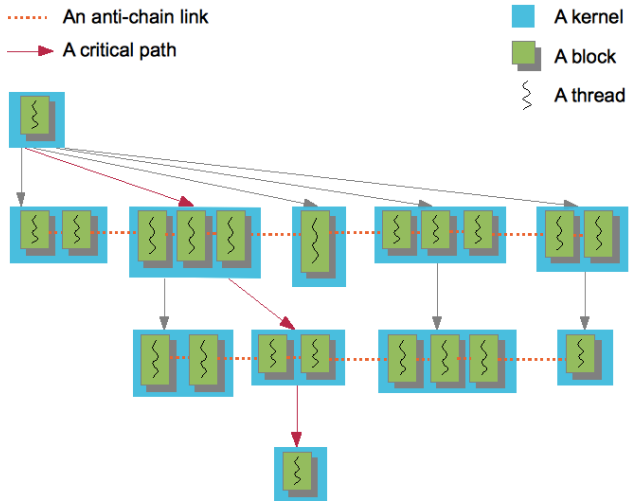


Work:

- The *work* $W(B)$ of a thread-block B is defined as the total number of local operations performed by the threads of B
- The *work* $W(K)$ of a kernel K is defined as the sum of the works of its thread-blocks
- The *work* $W(\mathcal{P})$ of the entire program $\mathcal{P}$ is defined as the total work of all its kernels

$$W(\mathcal{P}) = \sum_{K \in \mathcal{K}} W(K)$$

MMM: Complexity Measures



Span:

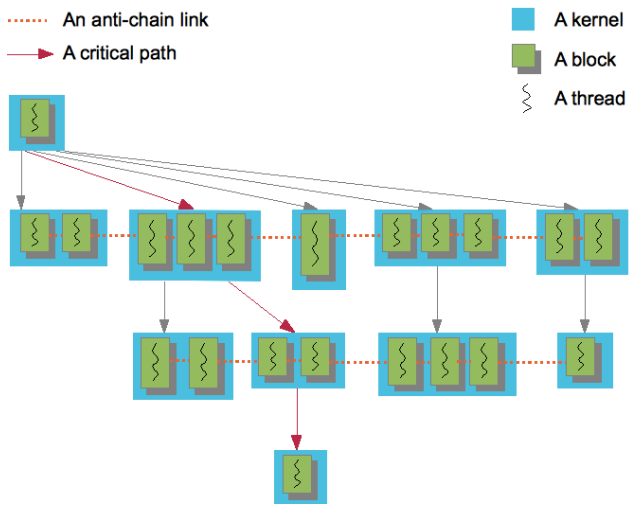
- The *span* $S(B)$ of a thread-block B is defined as the maximum number of local operations performed by a thread of B
- The *span* $S(K)$ of a kernel K is defined as the maximum span of its thread-blocks
- We define the span $S(\gamma)$ of any path γ from the first kernel to a last one as

$$S(\gamma) = \sum_{K \in \gamma} S(K)$$

- The *span* $S(\mathcal{P})$ of the entire program $\mathcal{P}$ is defined as

$$S(\mathcal{P}) = \max_{\gamma} S(\gamma)$$

MMM: Complexity Measures



Overhead:

- The *overhead* $O(B)$ of a thread-block B is defined as $(r + w)\,U$, assuming that each thread of B reads (at most) r words and writes (at most) w words to the global memory
- The *overhead* $O(K)$ of a kernel K is defined as the sum of the overheads of its thread-blocks
- The *overhead* $O(\mathcal{P})$ of the entire program $\mathcal{P}$ is defined as the total overhead of all its kernels

$$O(\mathcal{P}) = \sum_{\alpha} O(\alpha)$$

MMM: Graham-Brent Theorem with Overhead

Theorem (Graham-Brent)

We have the following estimate for the running time T_p of the program $\mathcal{P}$ when executed on p SMs

$$T_p \leq (N(\mathcal{P})/p + L(\mathcal{P})) \cdot C(\mathcal{P}),$$

where

- $N(\mathcal{P})$ *number of vertices in the thread-block DAG of* $\mathcal{P}$,
- $L(\mathcal{P})$ *critical path length (that is, the length of the longest path) in the thread-block DAG of* $\mathcal{P}$,
- $C(\mathcal{P})$ $= \max_{B \in \mathcal{B}(\mathcal{P})} (S(B) + O(B))$.

Algorithms Analysis and Optimization with MMM

- Polynomial Division Algorithms
- The Euclidean Algorithm for Polynominals
- Polynominal Multiplication Algorithms

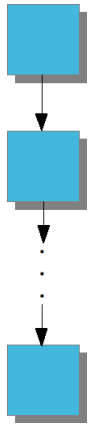
Plain Division Algorithms for Polynominals

Given two polynomials a and b over a finite field $\mathbb{K}$ and with variable $\mathbf{X}$, where $\deg(a) = n - 1$, and $\deg(b) = m - 1$, compute the remainder in the Euclidean division of a by b.

- Naive division algorithm
- Optimized division algorithm

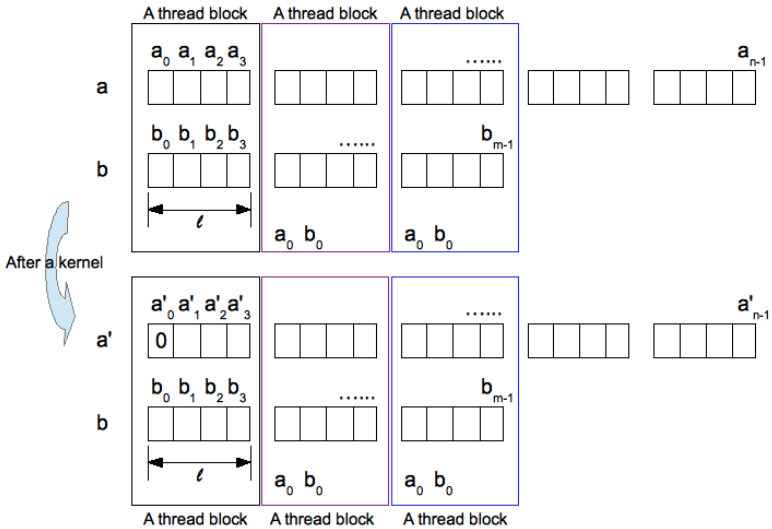
We assume that

- b is not zero
- $n \geq m$



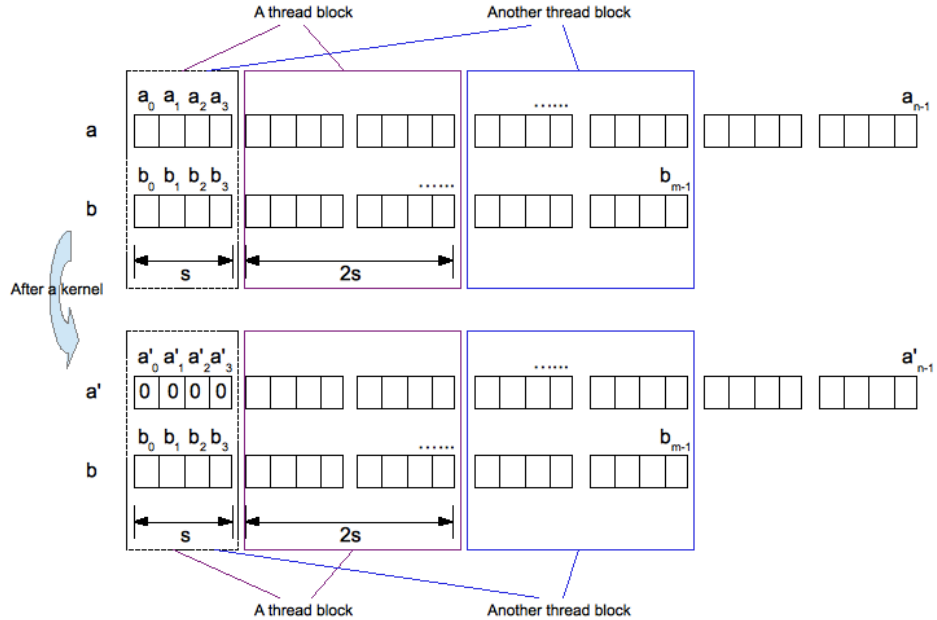
Plain Division Algorithms

Naive Division Algorithm



- Each kernel performs 1 division step
- $n - m + 1$ kernels are executed in serial

Optimized Division Algorithm



- Each kernel performs s division steps
- $\lceil \frac{n-m+1}{s} \rceil$ kernels are executed in serial

Analysis of Plain Division Algorithms

We obtain the work ratio and the overhead ratio as

$$\frac{W_{\text{nai}}}{W_{\text{opt}}} = \frac{8\,(Z+1)}{9\,Z+7} \quad \text{and} \quad \frac{O_{\text{nai}}}{O_{\text{opt}}} = \frac{20}{441}\,Z$$

Applying Theorem 1,

$$R = \frac{(N_{\text{nai}}/p + L_{\text{nai}}) \cdot C_{\text{nai}}}{(N_{\text{opt}}/p + L_{\text{opt}}) \cdot C_{\text{opt}}} = \frac{2}{3}\,\frac{(3+5\,U)\,(2\,m+Z\,p)\,Z}{(Z+21\,U)\,(7\,m+2\,Z\,p)}$$

When m escapes to infinity, the ratio R is equivalent to

$$\frac{4}{21}\,\frac{(3+5\,U)\,Z}{Z+21\,U}$$

- We observe that this latter ratio is larger than 1 if and only if $Z > \frac{441\,U}{20\,U-9}$ holds
- The optimized algorithm is overall better than the naive one

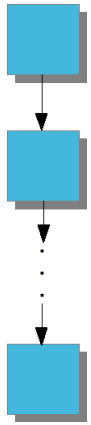
The Euclidean Algorithm for Polynominals

Given two polynomials a and b over a finite field $\mathbb{K}$ and with variable $\mathbf{X}$, where $\deg(a) = n-1$ and $\deg(b) = m-1$, compute the greatest common divisor (GCD) of a and b.

- Naive Euclidean Algorithm
- Optimized Euclidean Algorithm

We assume that

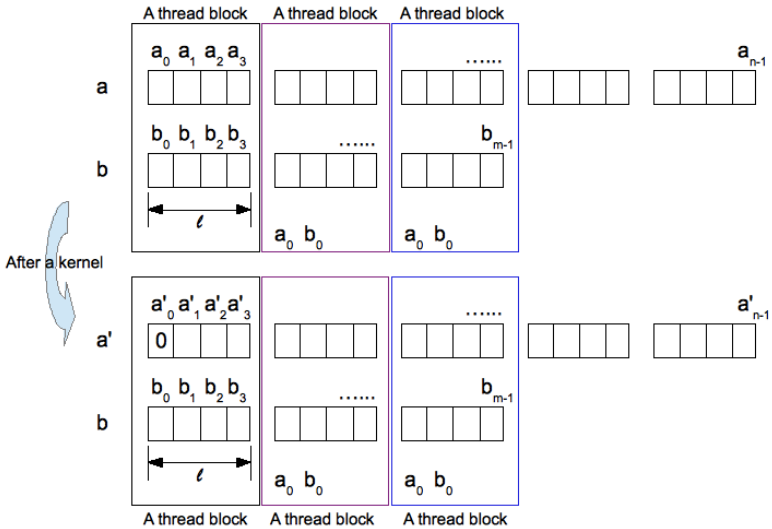
- b is not zero
- $n \geq m$



The Euclidean Algorithm

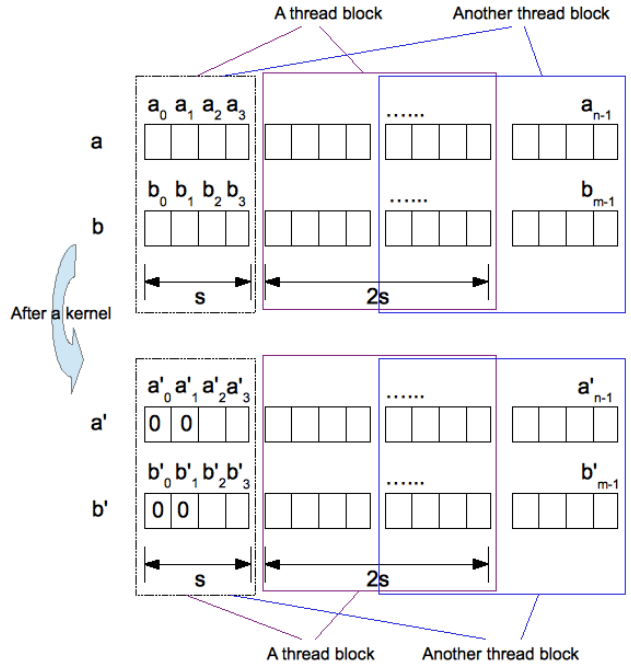
It checks the current degree of both a and b to decide which polynomial will take the role as a divisor, and then it completes a division step

Naive Euclidean Algorithm



- Each kernel performs 1 division step
- $n + m - 2$ kernels are executed in serial

Optimized Euclidean Algorithm



- Each kernel performs s division steps
- $\frac{n+m}{s}$ kernels are executed in serial

Analysis of the Euclidean Algorithm

We obtain the work ratio and the overhead ratio, replacing m by n as

$$\frac{W_{\text{nai}}}{W_{\text{opt}}} = \frac{(284\,Z+2)\,n^2+(Z-2)\,n}{(1296\,Z+7488)\,n^2+(348\,Z^2+2208\,Z)\,n-(115\,Z^3+616\,Z^2)}$$

$$\frac{O_{\text{nai}}}{O_{\text{opt}}} = \frac{5}{48}\,\frac{Z(2\,n+2+Z)}{6\,n+Z}$$

- As n escapes to infinity, the additional work $W_{\text{opt}} - W_{\text{nai}}$ is only a portion of W_{nai},
- Meanwhile the data transfer overhead decreases as Z increases.

Applying Theorem 1, when n escapes to infinity, the ratio R is equivalent to

$$R = \frac{(N_{\text{nai}}/p + L_{\text{nai}}) \cdot C_{\text{nai}}}{(N_{\text{opt}}/p + L_{\text{opt}}) \cdot C_{\text{opt}}} \simeq \frac{(3+5\,U)\,Z}{9\,(Z+16\,U)}$$

- We observe that this latter ratio is larger than 1 if and only if $Z > \frac{144\,U}{5\,U-6}$ holds
- The optimized algorithm is overall better than the naive one

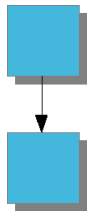
Analysis of (Plain) Polynominal Multiplication

Given two polynomials a and b over a finite field $\mathbb{K}$ and with variable $\mathbf{X}$, where $\deg(a) = n-1$ and $\deg(b) = m-1$, compute the product d of $a \times b$

1) Multiplication phase
2) Addition phase

We assume that

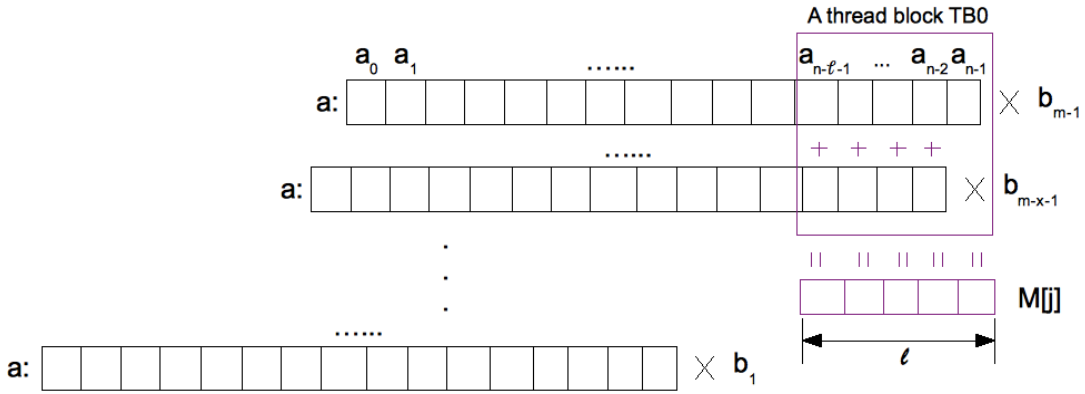
- $n \geq m$



Multiplication Phase

Algorithm 1: MulKer(a, b, M, n, x)

Input: $a, b, M \in \mathbb{K}[X]$ and an integer $x \geq 1$.
j = blockID×blockDim+threadID; t = threadID;
Let B' and A' be two local arrays of size x and blockDim respectively with coefficients in $\mathbb{K}$;
$i = j \mod (n + x - 1)$;
if $i \geq n$ **then**
  $A'[t] = 0$;
else
  $A'[t] = a[i]$;
if $t < x$ **then**
  $B'[t] = b[\lfloor (j/(n + x - 1)) \rfloor x + t]$;
/* copying from global */
$f = 0$;
for $(k = 0; k < x \wedge (i - k) \geq 0; k = k + 1)$ **do**
  $f = f + A'[i - k]\, B'[k]$;
$M[j] = f$;
/* writing to global memory */

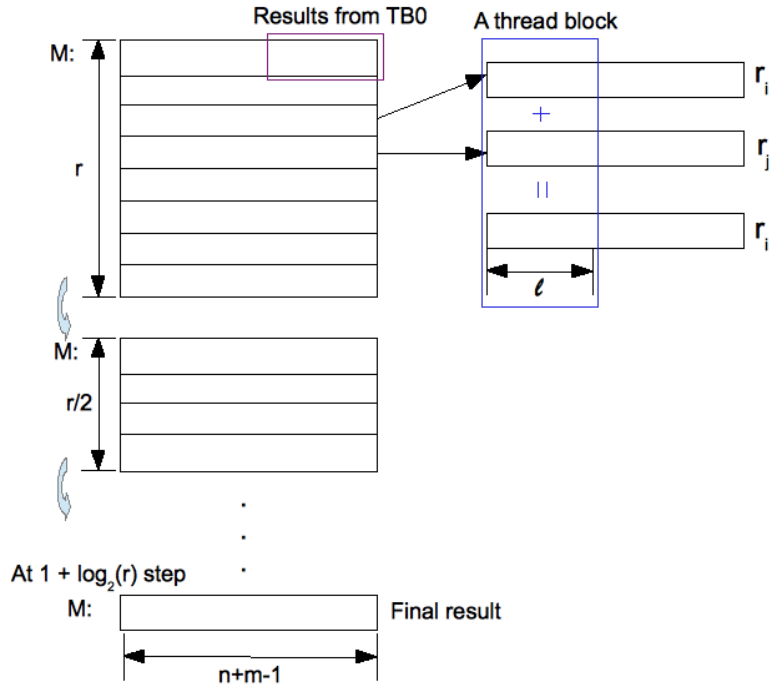


- Within a ℓ-thread block, (1) each thread reads a coefficient from a, (2) x threads read a coefficient from b and (3) ℓ partial sums are written to M

Addition Phase

Algorithm 2: AddKer(M, d, c, x, r, i)

Input: $M, d, \in \mathbb{K}[X]$ and c, x, r, i are positive integers.
j =blockID blockDim +threadID; t =threadID;
$k = j \mod c$;
$q = \lfloor j/c \rfloor$;
$s = 2^i - 1 + 2^{i+1} q$;
$e = s + 2^i$;
if $k < 2^i x$ **then**
  $d[s x + k] = d[s x + k] + M[s r + k]$;
else
  $M[e r + k - 2^i x] = M[e r + k - 2^i x] + M[s r + k]$;



- Each thread block needs ℓ intermediate results from two rows of M, and ℓ intermediate results to write back

Analysis of the Polynominal Multiplication Algorithm

Arbitrary x

- We have work, span and overhead as

$$\begin{aligned} W_{\mathrm{x}} &= (2\,m - \tfrac{1}{2})\,(n + x - 1) \\ S_{\mathrm{x}} &= 2\,x - 1 + \log_2 \frac{m}{x} \\ O_{\mathrm{x}} &= \frac{3\,(n+x-1)\,(2\,m-x)\,U}{x\,\ell} \end{aligned}$$

- To apply Theorem 1, we have

$$\begin{aligned} N_{\mathrm{x}} &= \frac{(n+x-1)\,(2\,m-x)}{x\,\ell} \\ L_{\mathrm{x}} &= \log_2 \frac{m}{x} + 1 \\ C_{\mathrm{x}} &= 2\,x - 1 + 3\,U \end{aligned}$$

Comparison of Polynominal Multiplication Algorithms

We replace x by 1 to obtain a "naive" algorithm. Then, we obtain the work ratio and the overhead ratio as

$$\frac{W_1}{W_x} = \frac{n}{n+x-1} \quad \text{and} \quad \frac{O_1}{O_x} = \frac{n\,(2\,m-1)\,x}{(n+x-1)\,(2\,m-x)}$$

Applying Theorem 1 and replacing m by n, when n escapes to infinity, the ratio R is equivalent to

$$R = \frac{(N_1/p + L_1)\cdot C_1}{(N_x/p + L_x)\cdot C_x} \simeq \frac{(1+3\,U)\,x}{2\,x-1+3\,U}$$

- One can assume $3\,U > 1$, which implies that the above ratio is always greater than 1 as soon as $x > 1$ holds
- The algorithm with arbitrary x outperforms the naive one

Conclusions

- We have introduced a model of multithreaded computation, combining the fork-join parallelism and the SIMD parallelism.
- Our goal is to reduce scheduling and communication costs in GPU programs.
- Three applications, for which an implementation report can be found in [1], illustrate the effectiveness of our model.
- For each of these three applications,
  - we determined a program parameter so as to minimize parallelism overhead
  - we checked that the work overhead introduced by this optimization technique remains very low.
  - we estimated the running time on p SMs with an enhanced version of Graham-Brent theorem..

[1] S. A. Haque and M. Moreno Maza, "Plain polynomial arithmetic on GPU," in J. of Physics: Conf. Series, vol. 385, no. 1. IOP Publishing, 2012, p. 12014.