On the Parallelization of Subproduct Tree Techniques Targeting Many-core Architectures

Sardar Anisul Haque, Farnam Mansouri, Marc Moreno Maza

University of Western Ontario, Canada

HPCS 2013 – University of Ottawa
June 18, 2013

Content

- Problem Definition
- Subproduct Tree
- Implementation
- Experimentation
- Conclusion
- Reference

Problem Definition

Polynomial Interpolation

Given distinct points $u_0,\ u_1, \ldots, u_{n-1}$ in a field K and arbitrary values $v_0,\ v_1, \ldots, v_{n-1} \in K$, compute the unique polynomial $P \in K[x]$ of degree less than n that takes the value v_i at the point u_i for all i. For convenience, we will assume that $n = 2^k$ holds for some k.

$$\boxed{((u_0, v_0), \ldots, (u_{n-1}, v_{n-1}))} \xrightarrow{?} \boxed{P \text{ where } P(u_i) = v_i \text{ for } 0 \leq i < n}$$

- Application: polynomial system solvers (numerical and symbolic), cryptography, etc.
- Advantages: creates opportunities to use asymptotically fast algorithms (FFT-based) and concurrent execution.
- Rationale: FFT-based techniques require special evaluation points (consecutive powers of a primitive root of unity).
- Work-around: using subproduct-tree techniques relax this latter point, but are hard to parallelize!

Subproduct Tree

Polynomial Interpolation (Recall)

Given distinct points u_0, $u_1, \dots, u_{n-1}$ in a field K and arbitrary values v_0, $v_1, \dots, v_{n-1} \in K$, compute the unique polynomial $P \in K[x]$ of degree less than n that takes the value v_i at the point u_i for all i. For convenience, we will assume that $n = 2^k$ holds for some k.

Definition

- Split the point set $U = \{u_0, \dots, u_{n-1}\}$ into two parts of equal cardinality and proceed recursively with each part until each of them has only one element.
- This leads to a binary tree of depth k having the points $u_0, \dots, u_{n-1}$ as leaves.
- Let $m_i = x - u_i$ and for $0 \leq i \leq k$ and $0 \leq j < 2^{k-j}$ define

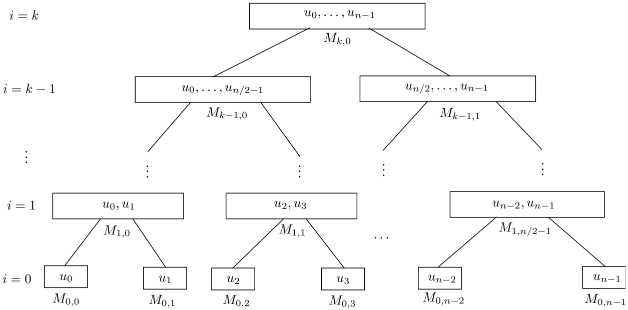
$$M_{i,j} = m_{j \cdot 2^i} \cdot m_{j \cdot 2^i + 1} \dots m_{j \cdot 2^i + (2^i - 1)} = \prod_{0 \leq l < 2^i} m_{j \cdot 2^i + l}$$

Subproduct Tree

Definition (Recall)

Let $m_i = x - u_i$ and for $0 \leq i \leq k$ and $0 \leq j < 2^{k-j}$ define

$$M_{i,j} = m_{j\cdot 2^i} \cdot m_{j\cdot 2^i+1} \dots m_{j\cdot 2^i+(2^i-1)} = \prod_{0\leq l<2^i} m_{j\cdot 2^i+l}$$



Subproduct Tree

Applications

- Once the subproduct tree on $U = \{u_0, \ldots, u_{n-1}\}$ is computed,
- then one can evaluate $f \in K[x]$ of degree $n-1$ at every $u \in U$ essentially in linear time (up to log factors) thanks to FFT.
- One can also interpolate on U with values $V = v_0, \ldots, v_{n-1}\}$ essentially in linear time (up to log factors).
- All the corresponding algorithms are divide-and-conquer and reduce all arithmetic computations to polynomial multiplication.

Challenges toward parallelization

- This d-n-c formulation does not provide enough parallelism
- Thus one must also parallelize polynomial multiplication.
- Algorithms based on FFT (such as subproduct tree techniques) have ratio of work to memory access which is essentially constant, thus not well suited for multi-core architectures.
- During the execution of subproduct tree operation work load of the tasks varies greatly, not well suited for many-core GPUs.

Contributions of this work

Summary

- We propose parallel algorithms for performing subproduct tree construction, evaluation and interpolation and report on their implementation on many-core GPUs
- We enhance the traditional algorithms for polynomial evaluation and interpolation based on subproduct-tree trees, by introducing the notion of a subinverse tree.
- For subproduct-tree operations, we demonstrate the importance of adaptive algorithms. That is, algorithms that adapt their behavior to the available computing resources.
- In particular, we combine parallel plain arithmetic and parallel fast arithmetic.

FFT-based multiplication

Computing $H = FG$

- Let $F, G \in K[x]$ with degree $\frac{n}{2} - 1$.
- Compute $DFT(F, n), DFT(G, n)$ using a n-th primitive root of unity.
- Perform point-wise multiplication of the vectors $DFT(F, n), DFT(G, n)$
- Obtain H by inverse FFT.
- Algebraic complexity $= c_1 n \log(n) + c_2 n$
- In theory $c_1 = 4.5$ and $c_2 = 4$
- In our implementation $c_1 = 15$ and $c_2 = 2$

Performance Issues

- High Algebraic Complexity for small n
- Constant ratio work to memory access, challenging for small n again.

Plain polynomial arithmetic

$$\underbrace{\boxed{m_{d-1}}\ \boxed{\cdots}\quad \boxed{m_0}}_{d = 2^{i-1}+1}\ \underbrace{\boxed{i_{d-1}}\ \boxed{\cdots}\quad \boxed{i_0}}_{d = 2^{i-1}}\quad \underbrace{\boxed{?}\quad \boxed{?}\quad \boxed{\cdots}\quad \boxed{?}\quad \boxed{?}}_{d = 2^{i}}$$

Kernel Specifications

- Runs in quadratic time, but can be parallelized efficiently as we saw this morning.
- We use in low degree, thus on thread block can do one multiplication in shared memory.

Subproduct Tree Construction

Adaptive Algorithm

Let H be a fixed integer with $1 \leq H \leq k$. We call *adaptive algorithm for computing* the subproduct tree M_n on U *with threshold* H the following procedure:

1. For each level $1 \leq h \leq H$, we compute the subproducts using plain multiplication.
2. Then, for each level $H + 1 \leq h \leq k$, we compute the subproducts using FFT-based multiplication.

Algebraic Complexity

$$\frac{15}{2} n \log_2(n)^2 + \frac{19}{2} n \log_2(n) + f(H)n$$

with $f(H) \in O(2^H + H^2)$.

Polynomial Evaluation

Polynomial Evaluating

1. $r_0 = P\ rem\ M_{k-1,0}$ and $r_1 = P\ rem\ M_{k-1,1}$
2. Recursively compute $r_0(u_0), \ldots, r_0(u_{n/2-1}), r_1(u_{n/2}), \ldots, r_1(u_{n-1})$

Adaptive Top Down Traversing

Do the remaindering of the polynomials over subproducts, we fix a threshold H:

1. $1 \leq h \leq H$: use plain arithmetic
2. $H+1 \leq h \leq k$: use fast division (through Newton iteration and subinverse tree)

Algebraic Complexity

$$15\, n \log_2(n)^2 + 49\, n \log_2(n) + f(H)n$$

with $f(H) \in O(2^H + H^2)$.

Using Subinverse Tree

What is Subinverse Tree?

For the subproduct tree $M_n := \mathsf{SubproductTree}(u_0, \ldots, u_{n-1})$, the corresponding *subinverse tree* InvM is a complete binary tree with the same height as M_n and such that, at level i of InvM contains an univariate polynomial $\mathsf{InvM}_{i,j}$ of degree $2^i - 1$ such that for all $0 \leq j < 2^{k-i}$. we have

$$\mathsf{InvM}_{i,j}\, \mathsf{rev}_{2^i+1}(M_{i,j}) \;\equiv\; 1 \mod x^{2^i}.$$

Remarks

1. It is used to speedup multi-point evaluation in the degrees where fast division (based on *Newton iteration*) applies.
2. However, subinverse tree is not used in lower degrees.

Algebraic Complexity

$$10n\log(n) + 30n\log(n)^2 + f(H)n$$

with $f(H) \in O(2^H + H^2)$.

Polynomial Interpolation

Lagrange interpolation

1. we have $((u_0, v_0), \ldots, (u_{n-1}, v_{n-1}))$
2. $m = \prod_{0 \le i < n} (x - u_i)$, $s_i = \prod_{i \neq j} 1/(u_i - u_j)$
3. $f = \sum_{i=0}^{n} v_i s_i m/(x - u_i)$

Note: $1/s_i = m'(u_i)$, $P = M_{k-1,0} P_0 + M_{k-1,1} P_1$

Adaptive Algorithm

For computing intermediate results, we fix a threshold H:

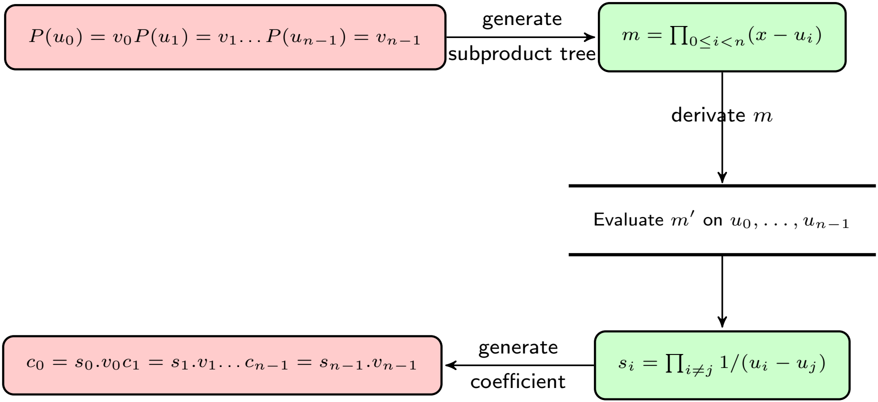
1. $1 \le h \le H$: use plain multiplication
2. $H + 1 \le h \le k$: use FFT-based multiplication

Algebraic Complexities

$$\frac{135}{2} n \log_2(n)^2 + \frac{177}{2} n \log_2(n) + f(H)n$$

with $f(H) \in O(2^H + H^2)$.

Lagrange Coefficients



$P(u_0) = v_0 P(u_1) = v_1 \ldots P(u_{n-1}) = v_{n-1}$

generate

subproduct tree

$m = \prod_{0 \le i < n} (x - u_i)$

derivate m

Evaluate m' on $u_0, \ldots, u_{n-1}$

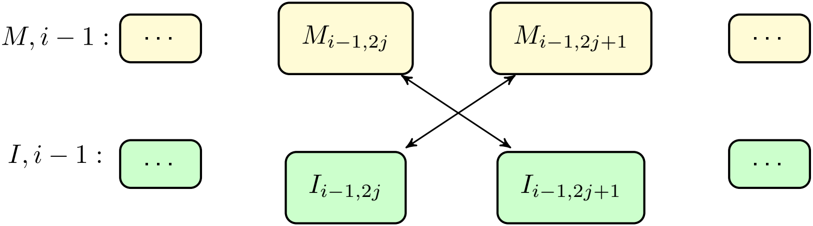
$s_i = \prod_{i \ne j} 1/(u_i - u_j)$

generate

coefficient

$c_0 = s_0 . v_0 c_1 = s_1 . v_1 \ldots c_{n-1} = s_{n-1} . v_{n-1}$

Linear Combination

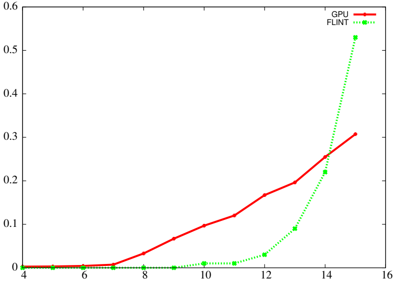


$$I_{i,j} = M_{i-1,2j} \times I_{i-1,2j+1} + M_{i-1,2j+1} \times I_{i-1,2j}$$

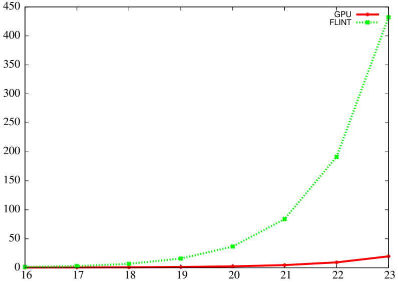
Experimentation, Runtimes

	Evaluation			Interpolation		
Deg.	GPU	FLINT	SpeedUp	GPU	FLINT	SpeedUp
8	0.0310	0	0	0.0328	0	0
9	0.0623	0	0	0.0669	0	0
10	0.0843	0	0	0.0968	0.01	0.1032
11	0.1012	0.01	0.0987	0.1202	0.01	0.0831
12	0.1361	0.02	0.1468	0.1671	0.03	0.1794
13	0.1580	0.07	0.4429	0.1963	0.09	0.4584
14	0.2034	0.17	0.8354	0.2548	0.22	0.8631
15	0.2415	0.41	1.6971	0.3073	0.53	1.7242
16	0.3126	0.99	3.1666	0.4026	1.26	3.1294
17	0.4285	2.33	5.4375	0.5677	2.94	5.1780
18	0.7106	5.43	7.6404	0.9034	6.81	7.5379
19	1.0936	12.63	11.5484	1.3931	15.85	11.3768
20	1.9412	29.2	15.0420	2.4363	36.61	15.0268
21	3.6927	67.18	18.1923	4.5965	83.98	18.2702
22	7.4855	153.07	20.4486	9.2940	191.32	20.5851
23	15.796	346.44	21.9321	19.6923	432.13	21.9441

Experimentation, Plots



(a) Lower Degrees



(b) Higher Degrees

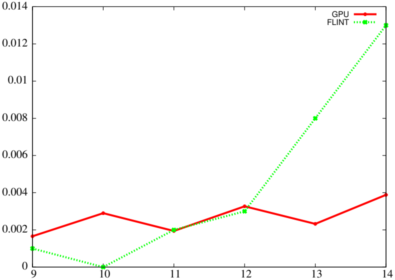
Effective Memory Bandwidth

Degree	Interpolation (GB/S)
10	0.1228
11	0.3403
12	0.7054
13	1.6182
14	3.1445
15	6.3464
16	11.4143
17	18.7800
18	26.7590
19	38.7674
20	49.0012
21	57.0978
22	62.4516
23	64.2464

Experimentation, Multiplications

Deg.	GPU	FLINT	SpeedUp
9	0.001	0.001	0.602
10	0.002	0	0
11	0.002	0.002	1.02
12	0.003	0.003	0.91
13	0.002	0.008	3.44
14	0.003	0.013	3.34
15	0.003	0.023	7.21
16	0.006	0.045	6.94
17	0.008	0.088	10.47
18	0.012	0.227	18.46
19	0.019	0.471	23.73
20	0.026	1.011	27.58
21	0.071	2.086	29.03
22	0.145	4.419	30.45
23	0.304	9.043	29.71

(c) Execution Times of Multiplication (s)



(d) Our GPU implementation versus FLINT

Conclusion

- First successfull parallelization of subproduct tree techniques
- Using adapting algorithm
- Implementating plain arithmetic
- Our polynomial multiplication in the range (2^9 to 2^{13}) can still be improved.
- References on next page

References

- J. Gathen and J. Gerhard. Modern Computer Algebra. Cambridge University Press, 1999.
- S. A. Haque and M. Moreno Maza. Plain polynomial arithmetic on GPU. In J. of Physics: Conf. Series, volume 385, page 12014. IOP Publishing, 2012.
- William Hart, Fredrik Johansson, and Sebastian Pancratz. Flint fast library for number theory, 2011.
- M. Moreno Maza and W. Pan. Fast polynomial arithmetic on a GPU. J. of Physics: Conference Series, 256, 2010.
- J. Nickolls, I. Buck, M. Garland, and K. Skadron. Scalable parallel programming with CUDA. Queue, 6(2):4053, 2008.