Подходы к распараллеливанию

Операция умножения в поле чисел

ООО «Сайфер ЛТД», к.т.н. Влад Ковтун
Андрей Охрименко

Содержание

- Актуальность
- Существующие алгоритмы
- Общее описание подхода
- Распараллеливание 2x
- Распараллеливание Mx
- Сравнение производительности
- Выводы

Введение

Цель: повышение производительности операции умножения целых чисел с отложенным переносом посредством распараллеливания

Объект: процесс умножения целых чисел

Предмет: механизм переноса переполнения из старшего разряда и распараллеливания

Актуальность

Алгоритмы умножения

- «В столбик»
- Рекурсивный Карацубы-Офмана*
- Шёнхаге — Штрассена (Дискретное преобразование Фурье)
- Алгоритм Фюрера (развитие Шёнхаге — Штрассена)
- Comba*
- и другие

* Применимы для криптографических задач

Подходы к повышению производительности

- Увеличение разрядности машинных слов
- Использование специализированных потоковых команд процессоров (MMX, SSE, SSE2, SSE3, SSE4, SSE5)
- Распараллеливание (многопоточность)
- Совершенствование алгоритмов
- Совершенствование структур данных

Аппаратное обеспечение

- Процессоры общего назначения (CPU)
  - Intel (Xeon 7000 Series: 10 ядер, 20 потоков)
  - AMD (Opteron 6200 Series: 16 ядер, 16 потоков)
- Графические процессоры общего назначения (GP GPU)
  - Nvidia (Tesla Fermi M2090: 512 ядер)
  - AMD (FireStream 9370: 1600 потоковых ядер + 20 SIMD процессоров)

Языки, библиотеки и технологии

- Библиотеки
  - Системные вызовы ОС (CPU)
  - Open MP (CPU)
  - Intel Threading Building Block (CPU)
- Язык
  - Open CL (CPU, GP GPU)*
  - CAL/IL (GP GPU AMD)
  - DirectX 11 – DirectCompute (GP GPU для Windows)
- Технологии
  - Nvidia CUDA (GP GPU)
  - AMD Accelerated Parallel Processing (GP GPU)

*Поддержка Nvidia CUDA, AMD APP

Усовершенствования

- Избавиться от последовательного переноса – изменение порядка вычислений:
  - Использование 32-х и 64-х данных для накопления переносов
  - Применение переносов по необходимости
- Распараллелить не связанные вычисления:
  - Цикл накопления суммы произведений

Алгоритм Comba

Алгоритм Comba. Умножение целых

Вход: целое $a, b \in \mathbf{GF}(p)$, $w = 32$, $n = \log_{2^w} a$, $nk = 2n - 1$.

Выход: $c = a \cdot b$

1. $r_0^{(32)} \leftarrow 0$, $r_1^{(32)} \leftarrow 0$, $r_2^{(32)} \leftarrow 0$.

2. For $k \leftarrow 0$, $k < n$, $k{+}{+}$ do

2.1. For $i \leftarrow 0$, $j \leftarrow k$, $i \le k$, $i{+}{+}$, $j{-}{-}$ do

2.1.1. $(uv)^{(64)} \leftarrow a_i^{(32)} \cdot b_j^{(32)}$.

2.1.2. $r_0^{(32)} \leftarrow r_0^{(32)} + v^{(32)}$, $r_1^{(32)} \leftarrow r_1^{(32)} + u^{(32)} + carry$, $carry \leftarrow 0$.

2.1.3. $r_2^{(32)} \leftarrow r_2^{(32)} + carry$, $carry \leftarrow 0$.

2.2. $c_k^{(32)} \leftarrow r_0^{(32)}$, $r_0^{(32)} \leftarrow r_1^{(32)}$, $r_1^{(32)} \leftarrow r_2^{(32)}$, $r_2^{(32)} \leftarrow 0$.

3. For $k \leftarrow n$, $l \leftarrow 1, k < nk$, $k{+}{+}$, $l{+}{+}$ do

3.1. For $i \leftarrow l$, $j \leftarrow k - l$, $i < n$, $i{+}{+}$, $j{-}{-}$ do

3.1.1. $(uv)^{(64)} \leftarrow a_i^{(32)} \cdot b_j^{(32)}$.

3.1.2. $r_0^{(32)} \leftarrow r_0^{(32)} + v^{(32)}$, $r_1^{(32)} \leftarrow r_1^{(32)} + u^{(32)} + carry$, $carry \leftarrow 0$.

3.1.3. $r_2^{(32)} \leftarrow r_2^{(32)} + carry$, $carry \leftarrow 0$.

3.2. $c_k^{(32)} \leftarrow r_0^{(32)}$, $r_0^{(32)} \leftarrow r_1^{(32)}$, $r_1^{(32)} \leftarrow r_2^{(32)}$, $r_2^{(32)} \leftarrow 0$.

4. $c_{nk}^{(32)} \leftarrow r_0^{(32)}$.

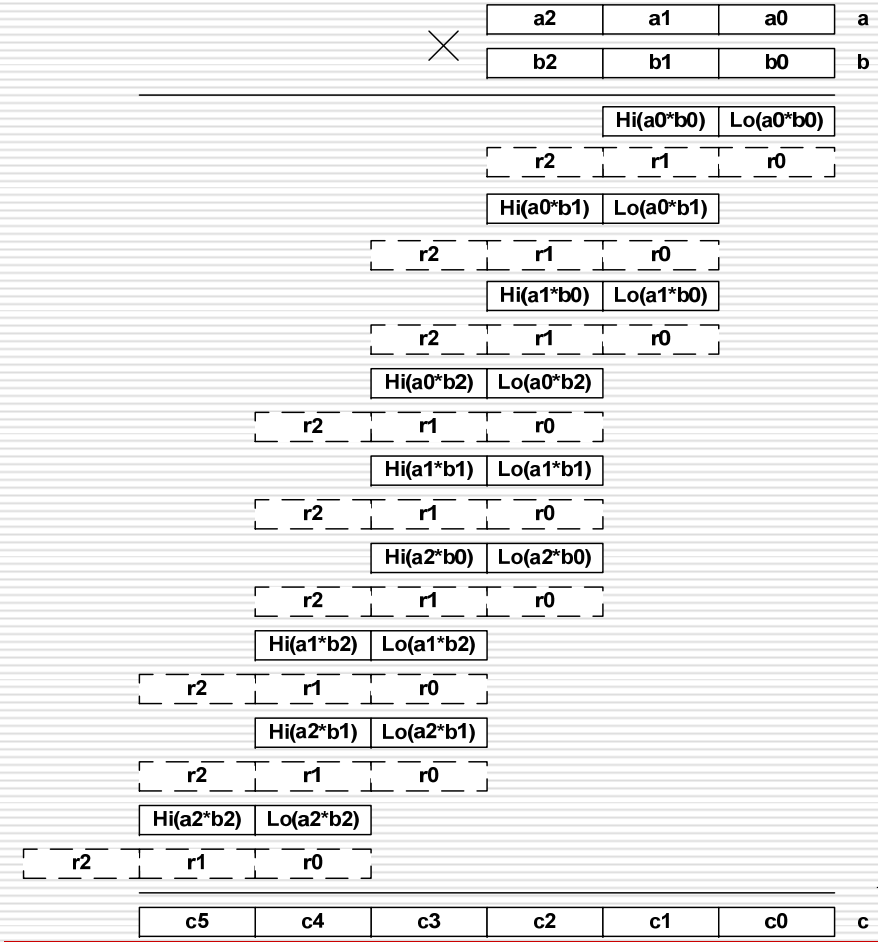
5. Return (c).

Алгоритм Comba: Сложность

$$I_{mul}^{Comba} = 4I_{assign}^{32} + n^2\left(1I_{mul}^{32} + 3I_{add}^{32} + 6I_{assign}^{32}\right) + 4\left(2n-1\right)I_{assign}^{32}$$

где I_{assign}^{32} - операция присвоения 32-х разрядных чисел, I_{add}^{32} - операция сложения 32-х разрядных чисел, I_{mul}^{32} - операция умножения 32-х разрядных чисел.

Алгоритм Comba: Структура



$$I_{mul}^{Comba} = 78I_{assign}^{32} + 9I_{mul}^{32} + 27I_{add}^{32}$$

Алгоритм Modified Comba

Алгоритм Modified Comba. Умножение целых

Вход: целое $a, b \in \mathbf{GF}(p)$, $w = 32$, $n = \log_{2^w} a$, $nk = 2n - 1$.

Выход: $c = a \cdot b$

1. $r_0^{(64)} \leftarrow 0$, $r_1^{(64)} \leftarrow 0$, $r_2^{(64)} \leftarrow 0$.

2. For $k \leftarrow 0$, $k < n$, $k{+}{+}$ do

2.1. For $i \leftarrow 0$, $j \leftarrow k$, $i \le k$, $i{+}{+}$, $j{-}{-}$ do

2.1.1. $(uv)^{(64)} \leftarrow a_i^{(32)} \cdot b_j^{32}$.

2.1.2. $r_0^{(64)} \leftarrow r_0^{(64)} + v^{(32)}$, $r_1^{(64)} \leftarrow r_1^{(64)} + u^{(32)}$.

2.2. $r_1^{(64)} \leftarrow r_1^{(64)} + \mathrm{hi}_{(32)}\left(r_0^{(64)}\right)$, $r_2^{(64)} \leftarrow r_2^{(64)} + \mathrm{hi}_{(32)}\left(r_1^{(64)}\right)$.

2.3. $c_k^{(32)} \leftarrow \mathrm{low}_{(32)}\left(r_0^{(64)}\right)$, $r_0^{(64)} \leftarrow \mathrm{low}_{(32)}\left(r_1^{(64)}\right)$, $r_1^{(64)} \leftarrow \mathrm{low}_{(32)}\left(r_2^{(64)}\right)$, $r_2^{(64)} \leftarrow 0$.

3. For $k \leftarrow n$, $l \leftarrow 1, k < nk$, $k{+}{+}$, $l{+}{+}$ do

3.1. For $i \leftarrow l$, $j \leftarrow k - l$, $i < n$, $i{+}{+}$, $j{-}{-}$ do

3.1.1. $(uv)^{(64)} \leftarrow a_i^{(32)} \cdot b_j^{32}$.

3.1.2. $r_0^{(64)} \leftarrow r_0^{(64)} + v^{(32)}$, $r_1^{(64)} \leftarrow r_1^{(64)} + u^{(32)}$.

3.2. $r_1^{(64)} \leftarrow r_1^{(64)} + \mathrm{hi}_{(32)}\left(r_0^{(64)}\right)$, $r_2^{(64)} \leftarrow r_2^{(64)} + \mathrm{hi}_{(32)}\left(r_1^{(64)}\right)$.

3.3. $c_k^{(32)} \leftarrow \mathrm{low}_{(32)}\left(r_0^{(64)}\right)$, $r_0^{(64)} \leftarrow \mathrm{low}_{(32)}\left(r_1^{(64)}\right)$, $r_1^{(64)} \leftarrow \mathrm{low}_{(32)}\left(r_2^{(64)}\right)$, $r_2^{(64)} \leftarrow 0$.

4. $c_{nk}^{(32)} \leftarrow \mathrm{low}_{(32)}\left(r_0^{(64)}\right)$.

5. Return (c).

Алгоритм Modified Comba 2x

Алгоритм Modified Comba 2x. Умножение целых с поддержкой OpenMP для 2-х потоков

Вход: целое $a, b \in \mathbf{GF}(p)$, $w = 32$, $n = \log_{2^w} a$, $nk = 2n - 1$.

Выход: $c = a \cdot b$

1. #pragma omp parallel sections begin

1.1. #pragma omp section begin

1.1.1. $rl_0^{(64)} \leftarrow 0$, $rl_1^{(64)} \leftarrow 0$, $rl_2^{(64)} \leftarrow 0$.

1.1.2. For $k \leftarrow 0$, $k < n$, $k{+}{+}$ do

1.1.2.1. For $i \leftarrow 0$, $j \leftarrow k$, $i \le k$, $i{+}{+}$, $j{-}{-}$ do

1.1.2.1.1. $(uv)^{(64)} \leftarrow a_i^{(32)} \cdot b_j^{32}$.

1.1.2.1.2. $rl_0^{(64)} \leftarrow rl_0^{(64)} + v^{(32)}$, $rl_1^{(64)} \leftarrow rl_1^{(64)} + u^{(32)}$.

1.1.2.2. $rl_1^{(64)} \leftarrow rl_1^{(64)} + \mathrm{hi}_{(32)}\left(rl_0^{(64)}\right)$, $rl_2^{(64)} \leftarrow rl_2^{(64)} + \mathrm{hi}_{(32)}\left(rl_1^{(64)}\right)$.

1.1.2.3. $c_k^{(32)} \leftarrow \mathrm{low}_{(32)}\left(rl_0^{(64)}\right)$, $rl_0^{(64)} \leftarrow \mathrm{low}_{(32)}\left(rl_1^{(64)}\right)$, $rl_1^{(64)} \leftarrow \mathrm{low}_{(32)}\left(rl_2^{(64)}\right)$, $rl_2^{(64)} \leftarrow 0$.

1.1.3. $r_0^{(64)} \leftarrow rl_1^{(64)}$.

1.1.4. $r_1^{(64)} \leftarrow rl_2^{(64)}$.

#pragma omp section end

1.2. #pragma omp section begin

1.2.1. $rl_0^{(64)} \leftarrow 0$, $rl_1^{(64)} \leftarrow 0$, $rl_2^{(64)} \leftarrow 0$.

1.2.2. For $k \leftarrow n$, $l \leftarrow 1$, $k < nk$, $k{+}{+}$, $l{+}{+}$ do

1.2.2.1. For $i \leftarrow l$, $j \leftarrow k - l$, $i < n$, $i{+}{+}$, $j{-}{-}$ do

1.2.2.1.1. $(uv)^{(64)} \leftarrow a_i^{(32)} \cdot b_j^{32}$.

1.2.2.1.2. $rl_0^{(64)} \leftarrow rl_0^{(64)} + v^{(32)}$, $rl_1^{(64)} \leftarrow rl_1^{(64)} + u^{(32)}$.

1.2.2.2. $rl_1^{(64)} \leftarrow rl_1^{(64)} + \mathrm{hi}_{(32)}\left(rl_0^{(64)}\right)$, $rl_2^{(64)} \leftarrow rl_2^{(64)} + \mathrm{hi}_{(32)}\left(rl_1^{(64)}\right)$.

1.2.2.3. $c_k^{(32)} \leftarrow \mathrm{low}_{(32)}\left(rl_0^{(64)}\right)$, $rl_0^{(64)} \leftarrow \mathrm{low}_{(32)}\left(rl_1^{(64)}\right)$, $rl_1^{(64)} \leftarrow \mathrm{low}_{(32)}\left(rl_2^{(64)}\right)$, $rl_2^{(64)} \leftarrow 0$.

#pragma omp section end

#pragma omp parallel sections end

2. $r_0^{(64)} \leftarrow r_0^{(64)} + c_n^{(32)}$. 3. $r_1^{(64)} \leftarrow r_1^{(64)} + \mathrm{hi}_{(32)}\left(rl_0^{(64)}\right) + c_{n+1}^{(32)}$.

4. $t^{(64)} \leftarrow \mathrm{hi}_{(32)}\left(rl_1^{(64)}\right)$.

5. For $k \leftarrow n + 2$, $k < nk$, $k{+}{+}$ do

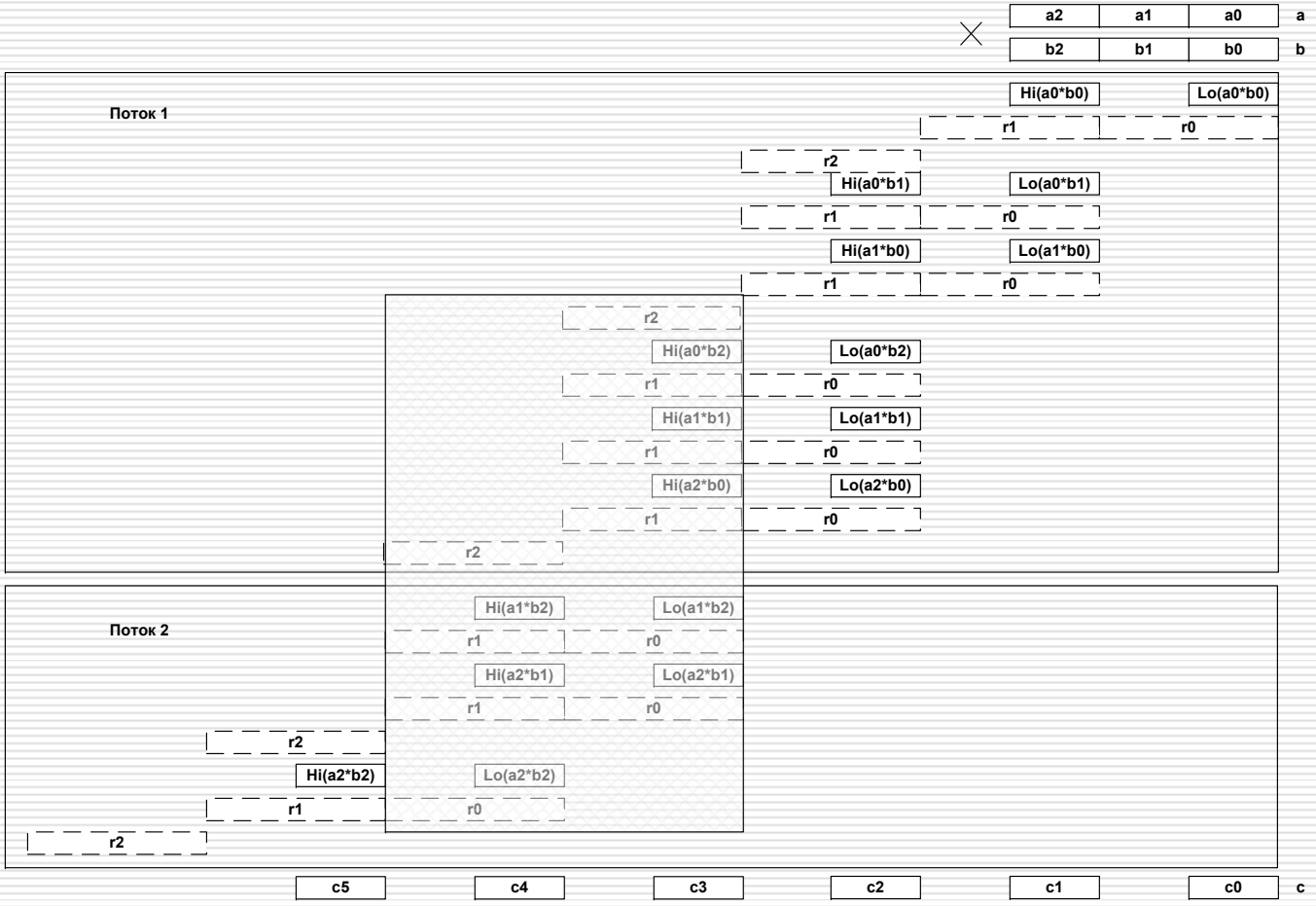
5.1. $t^{(64)} \leftarrow t^{(64)} + c_k^{(32)}$. $c_k^{(32)} \leftarrow \mathrm{low}_{(32)}\left(t^{(64)}\right)$.

5.2. $\mathrm{low}_{(32)}\left(t^{(64)}\right) \leftarrow \mathrm{hi}_{(32)}\left(t^{(64)}\right)$. $\mathrm{hi}_{(32)}\left(t^{(64)}\right) \leftarrow 0$.

6. $c_{nk}^{(32)} \leftarrow \mathrm{low}_{(32)}\left(r_0^{(64)}\right)$.

7. Return (c).

Алгоритм Modified Comba 2x



Алгоритм Modified Comba Mx

Алгоритм Modified Comba Mx. Умножение целых с поддержкой OpenMP для многопроцессорных систем

Вход: целое $a, b \in \mathbf{GF}(p)$, $w = 32$, $n = \log_{2^w} a$, $nk = 2n - 1$.

Выход: $c = a \cdot b$

1. #pragma omp parallel begin
2. #pragma omp for begin nowait

2.1. For $k \leftarrow 0$, $k < n$, $k{+}{+}$ do

2.1.1. $rl_0^{(64)} \leftarrow 0$, $rl_1^{(64)} \leftarrow 0$.

2.1.2. For $i \leftarrow 0$, $j \leftarrow k$, $i \le k$, $i{+}{+}$, $j{-}{-}$ do

2.1.2.1. $(uv)^{(64)} \leftarrow a_i^{(32)} \cdot b_j^{32}$.

2.1.2.2. $rl_0^{(64)} \leftarrow rl_0^{(64)} + v^{(32)}$, $rl_1^{(64)} \leftarrow rl_1^{(64)} + u^{(32)}$.

2.1.3. $r0_k^{(64)} \leftarrow rl_0^{(64)}$, $r1_k^{(64)} \leftarrow rl_1^{(64)}$.

#pragma omp for end

3. #pragma omp for nowait

3.1. For $k \leftarrow n$, $l \leftarrow 1, k < nk$, $k{+}{+}$, $l{+}{+}$ do

3.1.1. $rl_0^{(64)} \leftarrow 0$, $rl_1^{(64)} \leftarrow 0$, $rl_2^{(64)} \leftarrow 0$.

3.1.2. For $i \leftarrow l$, $j \leftarrow k - l$, $i < n$, $i{+}{+}$, $j{-}{-}$ do

3.1.2.1. $(uv)^{(64)} \leftarrow a_i^{(32)} \cdot b_j^{32}$.

3.1.2.2. $rl_0^{(64)} \leftarrow rl_0^{(64)} + v^{(32)}$, $rl_1^{(64)} \leftarrow rl_1^{(64)} + u^{(32)}$.

3.1.3. $r0_k^{(64)} \leftarrow rl_0^{(64)}$, $r1_k^{(64)} \leftarrow rl_1^{(64)}$.

#pragma omp for end

#pragma omp parallel end

4. $rl_0^{(64)} \leftarrow r0_0^{(64)}$.
5. $rl_1^{(64)} \leftarrow r1_0^{(64)}$.
6. $c_0^{(32)} \leftarrow \mathrm{low}_{(32)}\left(rl_0^{(64)}\right)$.
7. $rl_1^{(64)} \leftarrow rl_1^{(64)} + \mathrm{low}_{(32)}\left(rl_0^{(64)}\right)$.
8. $rl_2^{(64)} \leftarrow \mathrm{hi}_{(32)}\left(rl_1^{(64)}\right)$.
9. $rl_0^{(64)} \leftarrow rl_1^{(64)}$.
10. $rl_1^{(64)} \leftarrow rl_2^{(64)}$.
11. $rl_2^{(64)} \leftarrow 0$.
12. For $k \leftarrow 1$, $k < nk$, $k{+}{+}$ do

12.1. $rll_0^{(64)} \leftarrow r0_k^{(64)}$.

12.2. $rll_1^{(64)} \leftarrow r1_k^{(64)}$.

12.3. $rl_0^{(64)} \leftarrow rl_0^{(64)} + \mathrm{low}_{(32)}\left(rll_0^{(64)}\right)$.

12.4. $rl_1^{(64)} \leftarrow rl_0^{(64)} + \mathrm{hi}_{(32)}\left(rl_0^{(64)}\right) + \mathrm{hi}_{(32)}\left(rll_0^{(64)}\right) + \mathrm{low}_{(32)}\left(rll_1^{(64)}\right)$.

12.5. $rl_2^{(64)} \leftarrow rl_2^{(64)} + \mathrm{hi}_{(32)}\left(rl_1^{(64)}\right) + \mathrm{hi}_{(32)}\left(rll_1^{(64)}\right)$.

12.6. $c_k^{(32)} \leftarrow \mathrm{low}_{(32)}\left(rl_0^{(64)}\right)$. $rl_0^{(64)} \leftarrow rl_1^{(64)}$.

12.7. $rl_1^{(64)} \leftarrow rl_2^{(64)}$. $rl_2^{(64)} \leftarrow 0$.

13. $c_{nk}^{(32)} \leftarrow \mathrm{low}_{(32)}\left(rl_0^{(64)}\right)$.
14. Return (c).

Замеры производительности

- Тестовая программа на C++
- Скомпилированы Intel C++ Compiler XE 2011 с помощью Microsoft Visual Studio 2005 (Win32, Max Speed, SSE2)

Замеры производительности

Процессор	Операционная система	Ядер	Потоков выполнения команд	Примечание
Intel Dual Core T2130	Microsoft Windows 7 (x86)	2	2	
Intel Core2 Duo T7200	Microsoft Windows XP (x86)	2	2	
Intel Core2 Duo E6400	Microsoft Windows 7 (x64)	2	2	
AMD A83510 MX	Microsoft Windows 7 (x64)	4	4	
Intel Core i7-2600	Microsoft Windows 7 (x64)	4	8	With Hyper Threading

Замеры производительности

Процессор	Алгоритм	Количество 32-х битных слов / мс											
		3	4	8	16	32	48	64	96	128	192	256	384
Intel Dual Core T2130	Cmb*	16	16	46	187	702	1544	2652	5786	10246	22988	40206	90002
	Cmb* 2x	202	202	219	328	546	966	1591	3758	5381	13194	20508	49173
	Cmb* Mx	281	297	343	464	889	1544	2417	6332	8234	17935	33203	72519
Intel Core2 Duo T7200	Cmb*	16	16	46	156	484	1015	1734	3766	6719	14703	26063	57735
	Cmb* 2x	234	172	187	235	422	703	1125	2157	3641	7812	13531	29859
	Cmb* Mx	266	281	297	406	719	1235	1844	3593	6015	12891	22500	49625
Intel Core2 Duo E6400	Cmb*	0	16	31	94	328	688	1172	2515	4500	9843	17438	38610
	Cmb* 2x	78	93	93	125	266	453	719	1672	2438	5265	9547	20281
	Cmb* Mx	141	141	172	250	453	781	1218	2735	4047	8688	16000	34578
AMD A83510 MX	Cmb*	16	16	31	156	452	921	1576	3682	6537	14212	24133	51480
	Cmb* 2x	187	187	209	250	421	687	1061	2075	3416	7472	13072	28205
	Cmb* Mx	359	375	390	437	593	826	1185	2075	3276	6755	11404	24804
Intel Core i7-2600	Cmb*	0	16	16	78	249	546	936	2044	3525	5554	13884	30872
	Cmb* 2x	124	109	109	156	266	484	764	1622	2605	3635	9734	21902
	Cmb* Mx	172	156	203	234	312	421	609	1139	1794	7831	6334	13089

Выводы

- Эффект от использования Hyper-Threading значительный.
- Компиляторы показали плохие результаты производительности на Intel Core i7 2600:
  - Microsoft Visual C++ 2008 на Microsoft Windows 7 (x64)
  - Microsoft Visual C++ 2010 на Microsoft Windows 7 (x64)
  - GNU C++ на Linux Debian 6.0 (x64)
- Дальнейший интерес представляет перенос данного алгоритма на:
  - OpenCL (CPU, GP GPU)
  - Nvidia CUDA
  - AMD APP
- Распараллеливание остальных алгоритмов (add, sub, inverse, sqr, sqrt, exp) арифметических операций для многоядерных CPU, GP GPU

Выводы

- Существуют значительные накладные расходы на создание и настройку параллельных потоков, которые соизмеримы с решаемыми задачами

Вопросы?

Спасибо за внимание!

ООО «САЙФЕР ЛТД»

Владислав Ковтун

email: vlad.kovtun@cipher.kiev.ua
www: http://www.cipher.kiev.ua

Linux & GP GPU

- KGPU is a GPU computing framework for the Linux kernel. It allows the Linux kernel to directly execute CUDA programs running on GPUs. The motivation is to augment systems with GPUs so that like user-space applications, the operating system itself can benefit from the GPU acceleration. It can also offload computationally intensive work from the CPU by enabling the GPU as an extra computing device.
- The current KGPU release includes a demo task with GPU augmentation: a GPU AES cipher based eCryptfs, which is an encrypted file system on Linux. The read /write bandwidths are expected to be accelerated by a factor of 1.7 ~ 2.5 on an NVIDIA GeForce GTX 480 GPU.
- The source code can be obtained from https://github.com/wbsun/kgpu, and news and release information can be found at http://code.google.com/p/kgpu/.

* http://gpgpu.org