Опыт портирования многопоточного режима компилятора gcc на массивно-параллельную архитектуру отечественного процессора MALT

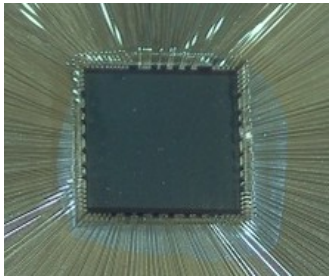
Кривов М.А.[1], Лукьянченко Г.А.[1], Роганов В.А.[1], Елизаров С.Г.[1,2]

МГУ имени М.В. Ломоносова

Переславль-Залесский, 27 ноября 2019

MALT: Manycore Architecture with Lightweight Threads

- Энергоэффективные процессоры с сотнями ядер на одном кристалле
- Разрабатываются в ЦИФ МГУ
- Архитектура адаптируется под задачи заказчика: MALT-C, MALT-D и MALT-F
- Имеются инженерные образцы (TSMC 28 нм)



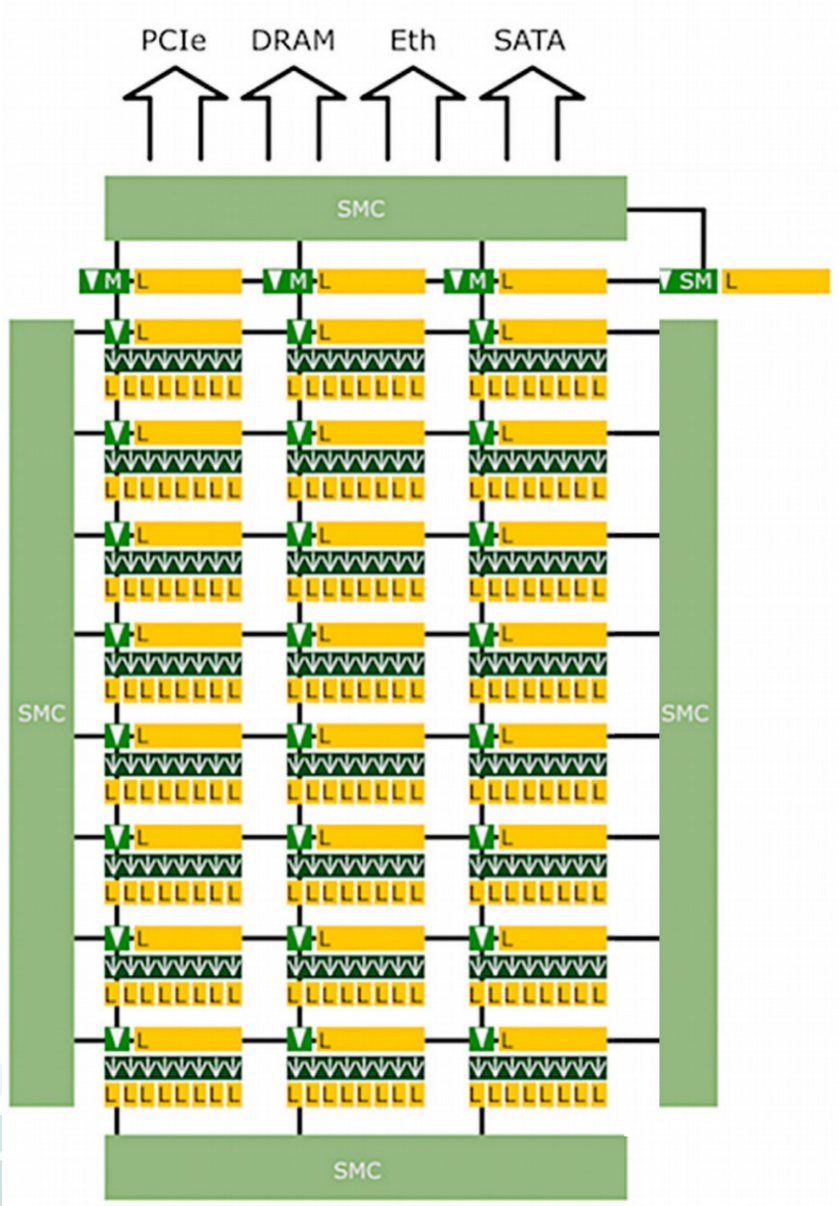
Кристалл 9Mb96G,
1-ое поколение

Демонстрационный
стенд

Подробнее на https://maltsystem.ru

MALT: Manycore Architecture with Lightweight Threads

- Рабочая частота до 1200 МГц
- До 256 универсальных ядер
- До 1024 векторных ядер
- До 8 Мбайт ОЗУ на кристалле
- До 96 Гбайт DDR3
- PCIe, 1Gb Eth, SATA
- TDP не более 50 Вт
- Производительность до 9.8 ТОп/с



Подробнее на https://maltsystem.ru

Неожиданные особенности

- Ради увеличения числа ядер пошли на следующие жертвы
  - Концепция *Bare-Metal*
    - одно приложение в один момент времени
    - операционная система как таковая отсутствует

Неожиданные особенности

- Ради увеличения числа ядер пошли на следующие жертвы
  - Концепция *Bare-Metal*
  - Нет кэшей данных L1/L2/L3, совсем нет

Неожиданные особенности

- Ради увеличения числа ядер пошли на следующие жертвы
  - Концепция *Bare-Metal*
  - Нет кэшей L1/L2/L3
  - Операции с F32/F64 реализуются программно
    - Но если очень нужно, то можно и аппаратно

Неожиданные особенности

- Ради увеличения числа ядер пошли на следующие жертвы
  - Концепция *Bare-Metal*
  - Нет кэшей L1/L2/L3
  - Операции с F32/F64 реализуются программно
  - Многоядерность, но не многопоточность
    - Ядро может переключиться на выполнение другого потока только программно

Неожиданные особенности

- Ради увеличения числа ядер пошли на следующие жертвы
  - Концепция *Bare-Metal*
  - Нет кэшей L1/L2/L3
  - Операции с F32/F64 реализуются программно
  - Многоядерность, но не многопоточность
  - Присутствует ядро-мастер, которое чуточку «равнее» остальных
    - Есть таймер, может использовать printf(), …
    - Особенности в основном затрагивают системные библиотеки

Неожиданные особенности

- **Ради увеличения числа ядер** пошли на следующие жертвы
  - Концепция *Bare-Metal*
  - Нет кэшей L1/L2/L3
  - Операции с F32/F64 реализуются программно
  - Многоядерность, но не многопоточность
  - Присутствует ядро-мастер, которое «равнее» остальных

Неожиданные особенности

- **Ради увеличения числа ядер** пошли на следующие жертвы
  - Концепция *Bare-Metal*
  - Нет кэшей L1/L2/L3
  - Операции с F32/F64 реализуются программно
  - Многоядерность, но не многопоточность
  - Присутствует ядро-мастер, которое «равнее» остальных

И приятные особенности

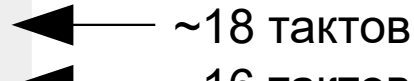
- При оптимизации ПО важно учитывать, что
  - Ядер очень много, а энергопотребление низкое
    - Например, 16 ядер при TDP ~3-5 Ватт
    - Можем разместить до 256 ядер (TSMC 28 нм)

И приятные особенности

- При оптимизации ПО важно учитывать, что
  - Ядер очень много, а энергопотребление низкое
  - Каждое ядро имеет быструю локальную память
    - Прямая адресация, используется для выделения объектов на стеке
    - Примерно 5 тактов на чтение/запись

```
1  // ...
2  float buf[256]
3  load_some_data(buf, 256);
4  float sum = 0;
5  for (int i = 0; i < 256; i++) {
6    sum += buf[i]
7  }
8  // ...
```

Если без регистров:



И приятные особенности

- При оптимизации ПО важно учитывать, что
  - Ядер очень много, а энергопотребление низкое
  - Каждое ядро имеет быструю локальную память
  - Имеются аппаратные мьютексы (Full/Empty Bits)
    - Каждая ячейка памяти снабжена флагом, указывающем на наличие значения
    - Если ячейка пуста, поток аппаратно блокируется, пока там что-то не появится

```
1  // ...
2  uint32_t *counter;
3  auto value = malt_read_modify(counter);
4  malt_fill_write(counter, value + 1);
5  // ...
```

А вот здесь FE-мьютекс

Кривов М.А., Елизаров С.Г., Реализация операций над B+ деревьями для массивно-параллельного процессора MALT-D

Оценка энергоэффективности

Процессор	Тех.проц., нм	Число ядер, шт	Частота, МГц	Производит., MOp/s	TDP, Ватт	Эффективн., MOp/(s*W)
MALT-D v1[1]	28	16	100	1,2	2	0,6
MALT-D v2[2]	20	96	100	8,1	11	0,73
Xeon X5680 x2[*]	32	6*2	3300	~110	130*2	0,42
Xeon E5-2666v3 x2[**]	22	10*2	2900	39,6	105*2	0,19

[1] Xilinx Kintex-7 xc7k325
[2] Xilinx Kintex UltraScale xcku115
[*] Sewall J. et al, PALM: Parallel Architecture-Friendly Latch-Free Modifications to B+ Trees on Many-Core Processors, 2011
[**] https://github.com/runshenzhu/palmtree

А что gcc?

- Поддерживает микроархитектуру ядра Microblaze, но не саму архитектуру MALT
- Компилирует код в режиме *single*, а не *posix*
- Это накладывает два ограничения:
  - Отключены *std::thread*, *std::mutex*, *std::atomic* и др.
  - Нет гарантии, что создаваемый код потокобезопасен
- Итог: работать можно, но неудобно

Потенциальные проблемы режима single

```
1   struct A {
2     A() { printf(«Greetings from A!»); }
3   }
4
5   int B() {
6     static A a1;
7     static A *a2 = new A();
8   }
9
10  // ...
11  malt_start_thread(B, 0, 0);
12  malt_start_thread(B, 0, 0);
13  // ...
```

Сломается или нет?

Включение режима posix

- Необходимо реализовать набор из 32 posix-подобных функций
- Указать соответствующий заголовочный файл *gthr-posix.h* при компиляции самого gcc
- При сборке пользовательской программы, слинковать её с реализацией этих функций

```
1  //...
2  int __gthread_create(__gthread_t *thread,
3                       void *(*func) (void*),
4                       void *args);
5
6  int __gthread_join(__gthread_t thread,
7                     void **value_ptr);
8
9  //...
```

Требуемые функции

- Работа с потоками
  - Create, Join, Detach, GetCurrent
- Работа с мьютексами
  - Create, Destroy, Lock, RecursiveLock, TryLock, TimedLock, UnLock
- Работа с ключами
  - Create, Destroy, Get, Set
- Работа с событиями
  - Create, Destroy, Wait, TimedWait, Wake, WakeAll
- Всё это требовалось реализовать поверх нативных функций MALT

… опускаем технические детали, рассматриваем наиболее интересные моменты …

```
Общий объём кода, Кбайт: 210
Общий объём кода, строк: 7218
Степень понятности:      чёрт ногу сломит
```

Замеры времени через такты

- На мастер-ядре есть таймер

```
1 return malt_get_time_us_64();
```

- На ядрах-рабочих подсчёт времени осуществляется через такты и частоты

```
1 malt_start_pcnt();
2
3 //...
4 auto freq = malt_get_core_freq();
5 uint64_t now;
6 malt_read_pcnt(0, 0, &now);
7 return (now * 1000) / freq;
```

Спинлок вместо мьютекса

- Хотя имеются аппаратные FE-мьютексы, они не поддерживают операции *TryLock*, *TimedLock*, *RecursiveLock*

```
1  void lock(mutex_object *mtx)
2  {
3    auto ptr = (volatile uint32_t *)&mtx->counter;
4
5    while (malt_ext_memop_inc(ptr, 1) != 2)
6    {
7      malt_ext_memop_inc(ptr, -1);
8      while (*ptr != 1) { YIELD(); }
9    }
10 }
```

Поток-диспечер на каждое ядро

- Функции MALT не позволяют реализовать POSIX-модель, поэтому есть поток-диспетчер:
  - Захватывает ядро при инициализации
  - Через интерконнект получает запросы на выполнение функций
  - При их отсутствии «засыпает» через FE-биты
  - Следит за коллекцией ключей *(требование POSIX)*
  - «Складирует» запросы, что позволяет обойти ограничение `число_потоков <= число_ядер`

И много другого

- Свой memory pool вместо malloc()
  - Учитывает особенности классов из std::
  - Требования от контроллера памяти
- Опции для настройки
  - Как «складировать» потоки
  - Сколько ядер задействовать и как размещать стэк
- Перевод всего, что возможно, на интерконнект и локальный стэк

Тестирование

- 70+ собственных тестов на базе фреймворка Google Tests

```
1  TEST(thread, ignore_thread_result)
2  {
3    ASSERT_TRUE(maltrt_create());
4    maltrt_thread_handle handle;
5    ASSERT_TRUE(maltrt_thread_create(&handle, ep,
6                                     nullptr));
7    ASSERT_TRUE(maltrt_thread_join(handle, nullptr));
8    ASSERT_TRUE(maltrt_destroy());
9  }
```

Тестирование

- 70+ собственных тестов на базе фреймворка Google Tests

```
[       OK ] parameter.wrong_stack_type (16 ms)
[ RUN      ] parameter.wrong_stack_size
c:\users\nn\documents\projects\ttglabs.com\projects\openmalt\m
 stack must be at least 1 KB
c:\users\nn\documents\projects\ttglabs.com\projects\openmalt\m
 stack must be at least 1 KB
[       OK ] parameter.wrong_stack_size (35 ms)
[----------] 23 tests from parameter (750 ms total)

[----------] Global test environment tear-down
[==========] 78 tests from 6 test cases ran. (2465 ms total)
[  PASSED  ] 78 tests.

C:\...bs.com\Projects\OpenMALT\malt_runtime\bin\Debug>_
```

Тестирование

- Стандартные тесты gcc (набор «30_threads»)

```
1  void test01() {
2    using namespace std;
3
4    future<void> f = async(Chucky{});
5    VERIFY( f.wait_for(chrono::seconds(100)) ==
6            future_status::deferred );
7
8    bool caught = false;
9    try {
10     f = async(launch::async, Chucky{});
11   } catch (const system_error&) {
12     caught = true;
13   }
14   VERIFY( caught );
15 }
```

Тестирование

- Стандартные тесты gcc (набор «30_threads»)

```
1  void test01() {
2    using namespace std;
3
4    future<void> f = async(Chucky{});
5    VERIFY( f.wait_for(chrono::seconds(100)) ==
6            future_status::deferred );
7
8    bool caught = false;
9    try {
10     f = async(launch::async, Chucky{});
11   } catch (const system_error&) {
12     caught = true;
13   }
14   VERIFY( caught );
15 }
```

Успешно проходят только ~90%

Замеры производительности

Настройки эмулятора:

- Количество ядер: 8
- Частота: 1 ГГц
- Локальная память: 16 КБ
- Задержка на чтение: 50 тактов
- Задержка на запись: 8 тактов

```
mc [mkrivov@ubuntu]:~/OpenMALT/hg/src/sw/examples/___
### Starting emulation (supermaster + 1 x 8 slaves), press Ctrl+E
to exit
### An emulator run using 1 host thread(s)
Boot v1.5 MALT 1.9.0A MALTemu

Benchmarking master core
Threads
   create
      C++11 (std):    1.278569e-06 seconds (2459 iterations)
      MALT Runtime:   7.320621e-07 seconds (3094 iterations)
      MALT Native:    9.120495e-07 seconds (9710 iterations)
   join
      C++11 (std):    2.171318e-06 seconds (2580 iterations)
      MALT Runtime:   1.803250e-06 seconds (3385 iterations)
      MALT Native:    N/A
   detach
      C++11 (std):    5.619785e-07 seconds (1678 iterations)
      MALT Runtime:   4.778606e-07 seconds (2281 iterations)
      MALT Native:    3.498786e-08 seconds (67109 iterations)
   current
      C++11 (std):    6.499830e-08 seconds (55863 iterations)
```

Замеры производительности (потоки)

Case	MALT Native	MALT Runtime	C++11
create()	4	8	16
join()	N/A	12	12
detach()	0	6	7
get_current()	0	0	1
many threads	48	243	337

Время выполнения с глобальным стэком (микросекунды)

Замедление отн. нативных функций (разы)

Case	MALT Native	MALT Runtime	C++11
create()	1	2,06	3,94
join()	N/A	N/A	N/A
detach()	1	69,87	71,41
get_current()	1	5,15	5,77
many threads	1	5,09	7,06

Case	MALT Native	MALT Runtime	C++11
create()	4,54	6,32	6,93
join()	N/A	5,01	3,62
detach()	1,06	8,86	8,94
get_current()	1	1,56	1,73
many threads	3,95	6,49	6,32

Ускорение от перехода на быстрый стэк (разы)

Замеры производительности (потоки)

Case	MALT Native	MALT Runtime	C++11
create()	4	8	16
join()	N/A	12	12
detach()	0	6	7
get_current()	0	0	1
many threads	48	243	337

Время выполнения с глобальным стэком (микросекунды)

Замедление отн. нативных функций (разы)

Case	MALT Native	MALT Runtime	C++11
create()	1	2,06	3,94
join()	N/A	N/A	N/A
detach()	1	69,87	71,41
get_current()	1	5,15	5,77
many threads	1	5,09	7,06

Case	MALT Native	MALT Runtime	C++11
create()	4,54	6,32	6,93
join()	N/A	5,01	3,62
detach()	1,06	8,86	8,94
get_current()	1	1,56	1,73
many threads	3,95	6,49	6,32

Ускорение от перехода на локальный стэк (разы)

Замеры производительности (мьютексы)

Замедление отн. нативных функций (разы)

Case	MALT Native	MALT Runtime	C++11
create()	1	1,84	1,84
destroy()	1	1,91	1,91
lock()	1	1,24	1,24
try_lock()	N/A	N/A	N/A
recursive_lock()	N/A	N/A	N/A
unlock()	1	3,72	3,72
many threads	1	2,94	3,01

Ускорение от перехода на локальный стэк (разы)

Case	MALT Native	MALT Runtime	C++11
create()	7,17	8,98	9,01
destroy()	7,72	9,17	8,85
lock()	11,02	8,76	8,53
try_lock()	N/A	7,71	7,64
recursive_lock()	N/A	2,31	2,27
unlock()	3,47	7,53	7,48
many threads	1	1	0,99

Заключение

- Разработанная библиотека MALT Runtime позволила перевести gcc в многопоточный режим. И всё работает!
- Деградация производительности составила от 2 до 7 раз (конструкции C++11 относительно нативных функций)
- Для дальнейшего развития требуется разрабатывать патч для классов из пространства std:: (пакет libsupc++)

Вопросы?

(m_krivov@cs.msu.su)