Динамическое распараллеливание для аппаратных ускорителей: некоторые алгоритмы и приложения

Владимир А. Роганов, НИИ Механики МГУ

Методы суперкомпьютерного моделирования,
г.Таруса, 03.10.2014

О чем доклад

Предмет исследования

- Динамическое распараллеливание программ ВНУТРИ CUDA[FPGA]-устройств
- Подходы к преобразованию “код->автогенератор параллельных потоков”

Модельное приложение

- Модельное приложение – интеллектуальная дешифровка паролей с целью оценки криптостойкости используемых алгоритмов
- Как проведенное исследование влияет на реальную жизнь

Примеры программных технологий, использующих динамическое распараллеливание

Языки программирования и их расширения

- SISAL
- Charm++
- Cilk
- CUDA (Compute Unified Device Architecture)
- T-система

Вариации/библиотеки на тему

- ThreadPool
- SuperCall

Пример: как пишут на T++

Материал из Википедии — свободной энциклопедии

T++ — язык программирования … расширяющий язык C++ несколькими словами, указывающими на возможность проведения параллельных вычислений:

```
tfun int fib(int n) {
  return n < 2 ? n : fib(n-1) + fib(n-2);
}
tfun int main (int argc, char *argv[]) {
  if (argc!=2) {printf("Usage: fib <n>\n"); return 1;}
  int n = atoi(argv[1]);
  printf("fib(%d) = %d\n", n, (int)fib(n));
  return 0;
}
```

Новые возможности CUDA 6.* [OpenCL 2.0]

CUDA 6.0 – 6.5 поддерживает:

- Общее адресное отображения для CPU и GPU
- Динамический параллелизм (запуск новых ядер из уже работающих ядер)
- Возможность использования GPU из нескольких параллельных потоков CPU

Проект OpenCL 2.0:

- Shared Virtual Memory
- Nested parallelism (выполнение kernel без привлечения host)
- pipe / atomic операции

Бонус: OpenCL может компилироваться для FPGA

Современные видеоадаптеры

	GEFORCE GTX 980
CUDA Cores	2048
Base Clock	1126 MHz
Boost Clock	1216 MHz
Single Precision	5 Teraflops
Memory Config	4GB / 256-bit GDDR5
Memory Speed	7.0 Gbps
Power Connectors	6-pin + 6-pin
TDP	165W

Как быть с реализацией динамического параллелизма на ускорителях

На аппаратных ускорителях далеко не всегда есть возможности явного сохранения и восстановления контекста, типа makecontext, swapcontext

Вопрос: нужен ли вообще DP на акселераторах?

Ответ: желателен - есть задачи, в которых гранулы параллелизма столь легкие, что накладные расходы на взаимодействие CPU<->GPU портят жизнь.

В конечном счете наиболее эффективно статико-динамическое распараллеливание программ.

Вариант №0: преобразование кода в конструкцию switch с помощью конвертера

```
class Fib : public Fn { // Fn has ‘st’ variable
  int n;
  TVar f1,f2;

  [...]

  if (n>1) {
    switch (st++) {
    case 0:
      f1=fib(n-1); f2=fib(n-2);
      WAIT_IF(needWait(f1)+needWait(f2));
    case 1:
      return f1+f2;
    }
  }

  [...]
```

Публикация в CUDA-Альманахе

NVidia неожиданно включила нашу статью в альманах

“Методы адаптации системы параллельного программирования OpenTS для поддержки работы T-приложений на гибридных вычислительных кластерах” в журнале Программные системы.

Алгоритм для (полу)автоматического динамического распараллеливания программ

Данный подход был придуман еще давно, но не находил применения в силу того, что он проигрывал другому подходу для случая CPU.

Однако практические результаты получены лишь недавно с использованием ресурсов ИПС РАН, в распоряжении которого имеется OpenTS (реализация языка T++) и современные ускорители NVidia (Tesla K20, GTX 750) с поддержкой динамического параллелизма.

Разделение дерева вызовов на уровни (внутренние и “листовые” вершины) + агрегация листовых вызовов.

Модельное приложение: обращение хэш-функций

Конкатенация пароля с солью - защита от радужных таблиц - неэффективна при наличии у злоумышленника мощной ЭВМ.

Движки сайтов (Joomla и т.д.) имеют уязвимости в своих расширениях, поддерживаемых случайными разработчиками. Утечка зашифрованных паролей через SQL-инъекцию приводит к серьезным проблемам.

Хорошие словари паролей

Множество реально используемых паролей имеет неслучайное распределение. Чем лучше мы будем учитывать “традиции” составления паролей (то есть свод правил для составления цепочек символов), тем эффективнее можно организовать перебор вариантов.

Общая схема алгоритма приложения md5t

Использование и синтез оценочных функций для паролей

- Вводится понятие цены цепочки символов
- Обучение “традиции” на основе заданного текста

Статико-динамическое распараллеливание

- Динамическое распараллеливание на кластерном уровне
- Предвычисленная таблица списков суффиксов
- Динамическое распределение счетных гранул по ядрам GPU

Код реализации MD5

```
void md5(uint8_t* msg, unsigned len, uint32_t h[4]) {
    uint32_t blk[16];
    memset(blk,0,64); memcpy(blk,msg,len);
    blk[14] = len<<3; ((uint8_t*)blk)[len] = 0x80;
    uint32_t a = h[0]=0x67452301, b = h[1]=0xefcdab89,
             c = h[2]=0x98badcfe, d = h[3]=0x10325476;
#define RT(a,b,expr,k,s,t) \
    a += (expr)+(t)+blk[k]; a = b+(a<<s|a>>(32-s));
#define R(a,b,c,d,k,s,t)  RT(a,b,d^(b&(c^d)),k,s,t)
    R(a, b, c, d,  0,  7, 0xd76aa478)
    R(d, a, b, c,  1, 12, 0xe8c7b756)
    R(c, d, a, b,  2, 17, 0x242070db)
    R(b, c, d, a,  3, 22, 0xc1bdceee)
    R(a, b, c, d,  4,  7, 0xf57c0faf)
    /////   Еще 70 строчек в том же духе  //////
#undef R
#undef RT
    h[0]+=a; h[1]+=b; h[2]+=c; h[3]+=d;
}
```

Полный код вычислительного CUDA-ядра

```
GBL void md5kernelDyn(GPU_Job::jb_t& b, HCmp* hcmp, bool
  const size_t id = blockIdx.x*blockDim.x + threadIdx.x;
  char s[11] = {0,};
  unsigned cnt = b.p[1]; //aligned cnt
  assert(!(cnt % THBLK));
  unsigned todo = cnt/THBLK;
  unsigned *p = b.p + 2; //skip r&a cnts
  for(int k=0;k<6;k++) s[k]=b.s12[k];
  for (int i=0; i<todo; i++) {
    uchar* sfx=(uchar*)(p+id*todo+i);
    for(int k=0;k<4;k++) s[k+6] = sfx[k];
    unsigned len=0;
    for (int k=0; k<10; k++) len+=(((char*)s)[len]!=0);
    uint32_t h[4] = { -1u, };
    md5((uint8_t*)s,len,h);
    int pi = (*hcmp)(h); //password index
    if (!(pi<0))
      { printf("PASSWORD[%d]: %s\n",pi,s); found = true;
  }
}
```

Пример запуска (1 видеокарта ценой $100)

```
Reading hash info
[0]: 69996ae2e822cc1f18404d4c66cc4932
.....................................
[11]: ea332c272f7e6498fbbdfe538feb9a1d
Got 12 hashes
PASSWORD[0]: adamovich
PASSWORD[1]: gmatveev
PASSWORD[2]: baranessa
PASSWORD[3]: biliberda
PASSWORD[9]: inikogda
PASSWORD[11]: moyparol
PASSWORD[8]: nizachto
PASSWORD[5]: parolchik
PASSWORD[7]: vslomati
```

Время счета: 65 минут

Преимущества и перспективы md5t

- Компактный, расширяемый код (< 500 строк)
- Произвольное комбинирование хэш-функций со строчными операциями
- Эффективная работа на гибридных кластерах (двухуровневое динамическое распараллеливание для CPU+GPU)
- Динамическая адаптация к языковым “традициям”
- Динамическое изменение функции-оценки и глубины счета в зависимости от успешности уже декодированных паролей

Выводы

1. CUDA 6.5 и GPU с capability > 3 реально обеспечивают качественно новые возможности для программирования. Реализация динамического распараллеливания при таких возможностях не является сложной задачей.
2. Ситуация с паролями сегодня критическая, как никогда! Злоумышленник при помощи новых видеоадаптеров и алгоритмов может в течении короткого времени обращать криптостойкие хэш-функции.
3. Необходимо СРОЧНО проводить модернизацию устаревших криптографических схем, используемых для работы с паролями.

Спасибо за внимание!

NVIDIA®

CUDA®

OpenCL