ISSN 2079-3316 ПРОГРАММНЫЕ СИСТЕМЫ: ТЕОРИЯ И ПРИЛОЖЕНИЯ №1(32), 2017, с. 173–191

УДК 519.682.3+519.245+519.644.7

С. М. Абрамов, В. А. Роганов, В. И. Осипов, Г. А. Матвеев

Метастохастические адаптивные алгоритмы и их реализация в супервычислительной среде T++&MPI

Аннотация. На примере предложенного адаптивного численного метода расчета многомерного определенного интеграла от априори неизвестной, вычислительно сложной функции, рассматривается общая методика и программный каркас для построения адаптивных параллельных решателей, нацеленных на интеллектуальное многовариантное моделирование сложных многопараметрических систем.

Ключевые слова и фразы: адаптивный алгоритм, параллельный алгоритм, метастохастический процесс, масштабируемое моделирование, многопараметрические системы, многомерный интеграл, метод Монте-Карло, рекурсивный спуск.

Введение

Создание адаптивных алгоритмов является на сегодняшний день одним из тех перспективных подходов, которые могут существенно увеличить производительность параллельных программ, используемых в науке и технике для моделирования различных процессов и явлений в неоднородных и динамично меняющихся непосредственно в процессе эволюции системах.

В отличие от традиционных подходов, в которых область решения разбивается на неизменные подобласти, а уравнение принято аппроксимировать и решать по фиксированной вычислительной схеме, адаптивные методы могут динамически выбирать способ декомпозиции на подзадачи и решатель для каждой из них, который учитывает возникшую специфику, связанную с неоднородностью получаемого решения и изменчивостью самих расчетных областей.

Работа выполнена в рамках проекта «Исследование и разработка методов решения ряда прикладных задач, использующих адаптивные алгоритмы" по программе фундаментальных исследований РАН I.5 на 2016 год "Проблемы создания высокопроизводительных распределенных и облачных систем и технологий. Интеллектуальные информационные технологии и системы».

Еще одной сильной стороной адаптивных алгоритмов является возможность эффективного учета изменчивости набора доступных вычислительных ресурсов. Если адаптироваться по областям и методам, то можно точно так же учитывать и набор имеющихся на данный момент счетных ядер. В случае использования распределенных вычислительных систем типа GRID это означает способность подключать новые рабочие узлы, а также обрабатывать их внезапное отключение с относительно небольшими потерями. Особенно важно это для современных суперЭВМ, количество счетных ядер в которых достигло такого количества, что отказы оборудования неизбежно возникают непосредственно уже во время счета. Адаптивные алгоритмы могут в этом случае произвести необходимые перевычисления для утраченных результатов и перестроить дальнейший счет с учетом отсутствия вышедших из строя узлов.

Наконец, появившаяся в последние годы ориентация на реконфигурируемые вычислительные суперЭВМ естественным образом увеличивает активность в области создания и реализации адаптивных алгоритмов, поскольку создает реальные возможности адаптировать аппаратный уровень под оптимальной метод решения для конкретной ситуации и стадии расчета.

Все вышеперечисленные факторы стимулируют активные исследования в указанной области, но они также и предъявляют дополнительные требования к используемой программной среде, в которой было бы несложно запрограммировать адаптивные алгоритмы на современных языка программирования и, что желательно, с использованием имеющихся и хорошо отлаженных блоков для решения различных традиционных подзадач, которые генерируются непосредственно во время работы адаптивного алгоритма. Требуется также поддержка динамического распараллеливания и отказоустойчивых вычислений.

Всеми необходимыми свойствами обладает среда программирования OpenTS, так как она изначально является реализацией подхода динамического распараллеливания и способна эффективно задействовать миллионы счетных ядер параллельных суперЭВМ, может работать на GRID/Cloud платформах и внутри современных графпроцессоров с поддержкой технологии CUDA. Краткое описание последних возможностей OpenTS можно найти в работе [1].

Одним из перспективных подходов к быстрой разработке адаптивных методов является переделка имеющегося традиционного параллельного кода для платформы MPI. Здесь для платформы OpenTS

имеются хорошие перспективы, поскольку она реализована поверх подсистемы DMPI (динамический MPI), и многие фрагменты параллельных программ для MPI удается заставить работать и в среде OpenTS. Некоторое падение производительности (порядка 10%) является при этом ожидаемым, но оно не идет ни в какое сравнение с тем выигрышем, который проявляется при адаптивной генерации отдельных подзадач, каждая из которых обрабатывается своим, специфическим и оптимальным для данной конкретной подзадачи, MPI-решателем, выбираемым в зависимости от возникающих во время счета условий. Такой подход (комплексирование кода на T++ и традиционного MPI-кода) был успешно протестирован, и результаты его апробации опубликованы в работе [2].

Важно отметить, что аналогичный подход для случая гибридной вычислительной платфомы «Кластер+CUDA» ранее был также успешно опробован, и результаты для этой платформы были описаны в работе [3].

Алгоритмы динамической генерации адаптивных сеток для языка T++ кратко рассматривались в работе [4].

Альтернативным подходом является разработка и реализация решающих ядер алгоритмов непосредственно на языке T++. В данном случае предполагается провести эксперименты вначале с несложными типовыми задачами, возникающими в разных численных методах, чтобы накопить необходимый опыт перед переходом к полноценным прикладным задачам и системам.

В данной статье рассмотрен алгоритм адаптивного интегрирования для априори-неизвестных (вычисляемых в процессе счета по мере необходимости) многомерных функций. Сама по себе задача многомерного интегрирования востребована во многих областях науки и техники, и в ней удается получить хорошие результаты при относительно несложной реализации на языке T++.

Основной идеей нового алгоритма адаптивного интегрирования является интеллектуальный анализ вычисляемых значений функции и такое рекурсивное разбиение интегрируемой области на подобласти, которое наиболее выгодно при имеющемся наборе счетных ядер, а также минимизирует прогнозируемое отклонение результата от точного значения.

Удобным средством при реализации рекурсивных алгоритмов на языке T++ является настраиваемый автоматический переход с T-вызовов на C-вызовы при достижении определенной глубины рекурсии. Это позволяет задействовать имеющееся множество ядер при

помощи вызовов T-версий функций, избегая накладных расходов на переключение контекста при низкоуровневом счете.

Следует также отметить, что отказоустойчивость разработанного расчетного кода для многомерного интегрирования обеспечивается базовыми возможностями среды OpenTS.

1. Метастохастические процессы и многостадийные вычисления

Введем сначала полезное понятие *метастохастического* процесса, начав с трактовки нового термина.

В русском языке под *стохастичностью* обычно подразумевают случайность, в то время как в греческом языке, откуда пришло это слово, оно имеет смысл «умеющий угадывать». Нетрудно заметить изначально позитивный смысл, вкладываемый в оригинальное значение.

В многостадийных вычислениях (multi-stage computations), которые сегодня широко используются в практике высокопроизводительных вычислений, отдельные стадии обычно представлены детерминированными процессами. Если же мы допустим, что некоторые из них являются случайными процессами, то получится достаточно интересный гибрид. В переводе с греческого языка слово μετάστοχαστικός можно интерпретировать как

«умеющий угадывать то, как надо угадывать»

2. Простая модельная задача для демонстрации подхода

Чтобы продемонстрировать идею многоуровневой организации стохастических процессов, возьмем простую модельную задачу — вычисление многомерного определенного интеграла.

Необходимость вычисления многомерных интегралов часто возникает в различных областях: в квантовой физике, в экономике.

Как известно, классические численные методы интегрирования типа прямоугольников и трапеций неэффективны при большом количестве измерений из-за экспоненциального роста необходимых вычислительных операций. Одним из решений, дающих неплохой результат, является использование вероятностных подходов типа методов Монте-Карло.

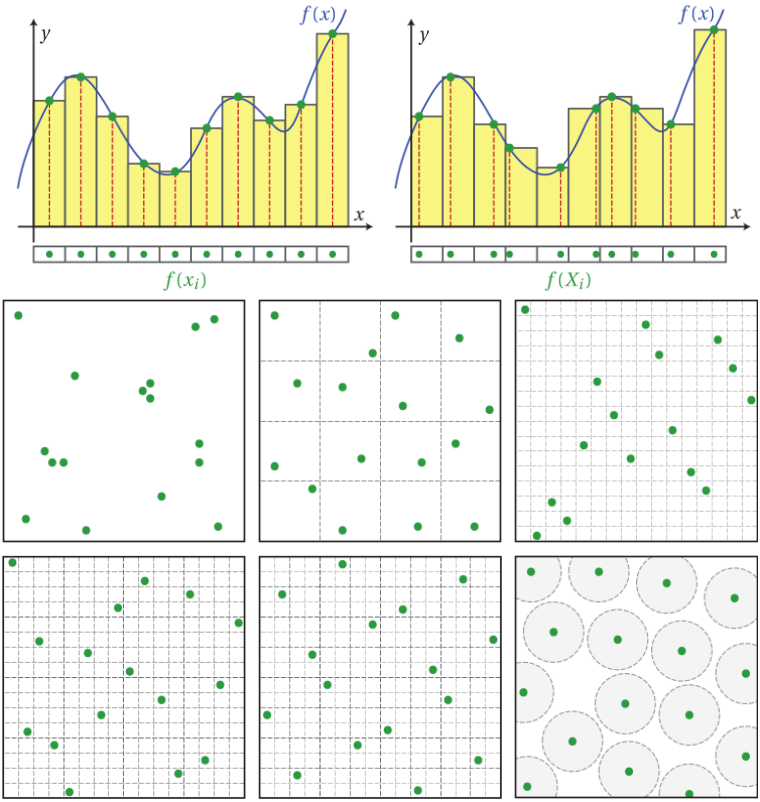

Рис. 1. Вычислительная схема расчета определенного интеграла

К сожалению, обычный метод Монте-Карло имеет медленную сходимость к точному значению. Поэтому существует большое количество модификаций метода, которые стараются при помощи разных ухищрений повысить точность при относительно небольшом количестве вычислений интегрируемой функции, поведение которой внутри интегрируемой области во многих случаях является априори неизвестным.

Для примера можно привести часто использованный стратифицированный метод и множество более экзотических, вычислительные принципы организации которых наглядно представлены на рис. 1.

3. Экспериментальный адаптивный алгоритм

Опишем кратко экспериментальный вариант метода Монте-Карло, в котором используются некоторые из вышеперечисленных идей.

Погрешность метода Монте-Карло при интегрирования пропорциональна вариации функции. Поскольку последняя нам не известна даже приближенно, логично пытаться оценить ее путем «вложенного» случайного процесса. Если для группы испытаний вариация полученного значения оказалась ниже некоторого порога, то можно просто усреднить полученные результаты счета. Иначе разумно разбить область по одному из измерений и продолжить процесс рассмотрения подобластей рекурсивно для каждой из полученных половинок.

Для простоты будем считать начальную область N-мерным гиперкубом. Полугиперкубом естественно называть его половинку (по любому измерению).

Введем понятие K-полугиперкуба. Будем называть 1-полугиперкубом полугиперкуб. Далее, если у нас имеется K-полугиперкуб, то $(K+1)$-полугиперкубом будет любая его половинка, отрезанная по еще не использованному для разрезания измерению.

Очевидно, что в N-мерном пространстве K может принимать значения от 1 до $N-1$. На шаге с номером N мы снова получим гиперкуб вдвое меньшего диаметра относительно начального.

K-полугиперкубы удобно использовать для разбиения области при интегрировании. Подробнее этот процесс описан в следующем разделе.

4. Экспериментальная реализация исследуемого адаптивного алгоритма

Для подкрепления правильности абстрактных идей и общих рассуждений конкретным была реализована простая экспериментальная программа общим объемом приблизительно 300 строк кода на языке T++, содержащая логику вышеописанного адаптивного варианта алгоритма вычисления многомерного определенного интеграла методом Монте-Карло.

Первая версия программы ориентирована на многоядерные процессоры, поэтому T-функции используют прямые ссылки в пределах единого адресного пространства (SMP-режим T-системы).

Ниже приведен код (с некоторыми сокращениями) с поясняющими комментариями.

Для начала определяются классы «отрезок» и «многомерный параллелепипед», которые используются для представления K-полугиперкубов:

T-System

```
struct Seg {
  double b,e;
  [...]
};
typedef unsigned dim_t;
struct PP {
  Seg seg[N];
  Seg& operator[] (dim_t i) {
    return seg[i];
  }
  [...]
  double volume() const {
    double v = 1;
    for (dim_t i = 0; i < N; i++)
      v *= seg[i].length();
    return v;
  }
  void splitBy(dim_t n, PP& pp0, PP& pp1) const {
    for (dim_t i=0; i < N; i++) {
      const Seg& s = seg[i];
      if (i != n) {
        pp0[i] = pp1[i] = s;
      } else {
        pp0[i].b = s.b;
        pp1[i].e = s.e;
        pp0[i].e = pp1[i].b = s.middle();
      }
    }
  }
}
```

Для хранения координат точек, в которых вычисляется интегрируемая функция, используется следующая структура данных:

T-System

```
struct Pt {
  double c[N];
  double vf;
  [...]
 Pt(const PP& pp) {
   random(pp);
 }
 void random(const PP& pp) {
   for (dim_t i=0; i<N; i++)
     c[i] = pp[i].random();
   vf = f(c);
```

```
    }
    bool inside(const PP& pp) const {
      for (dim_t i=0; i<N; i++)
        if (!pp[i].inside(c[i]))
          return false;
      return true;
    }
  };
```

Здесь метод *random* позволяет получать случайные точки внутри заданного многомерного параллелепипеда, а в методе *inside* проверяется их соответствующая принадлежность.

Класс *PtSet* используется для хранения наборов таких точек. Его дополнительный конструктор позволяет минимизировать количество вычислений функции f, если нами уже посчитан некоторый набор точек (он обычно известен для родительского K-полугиперкуба).

Метод *variation* класса *PtSet* подсчитывает вариацию для G различных приближений значения вычисляемого интеграла:

T-System

```
struct PtSet {
  unsigned n;
  Pt* p;
  [...]
  PtSet(unsigned _n, const PP& pp,
        const PtSet* ps = nullptr) : n(_n) {
    p = new Pt[n];
    unsigned j=0;
    for (unsigned i=0; i<n; i++) {
      bool set=false;
      if (ps != nullptr) {
        while (j < ps->n && !ps->p[j].inside(pp))
          j++;
        if (j < ps->n) {
          p[i] = ps->p[j++];
          set = true;
        }
      }
      if (!set)
          p[i].random(pp);
      }
    }
    double variation(double &average) const {
      double s[G];
      for (unsigned i=0; i<G; i++)
        s[i] = 0;
```

```
27          for (unsigned i=0; i<n; i++)
28            s[i % G] += p[i].vf;
29          for (unsigned i=0; i<G; i++)
30            s[i] /= n/G;
31          double vf=s[0], min=vf, max=vf, sum=vf;
32          for (unsigned i=1; i<G; i++) {
33            vf = s[i];
34            if (vf < min)
35              min = vf;
36            if (vf > max)
37              max = vf;
38            sum += vf;
39          }
40          average = sum / G;
41          return max - min;
42        }
43      };
```

Наконец, главный класс *GSCube* для представления K-полугиперкубов определяется следующим образом:

T-System

```
1   typedef int sign_t;
2     class GSCube : public PP {
3     public:
4       sign_t split[N];
5       [...]
6       void cubify() {
7         for (dim_t i=0; i<N; i++) {
8           assert(split[i]);
9           split[i] = 0;
10        }
11      }
12      dim_t availSplit() const {
13        dim_t i = 0;
14        while (i < N)
15          if (!split[i]) break; else i++;
16        return i;
17      }
18      void splitBy(dim_t n, GSCube& c0, GSCube& c1) const {
19           assert(!split[n]);
20           memcpy(c0.split,split,sizeof(split));
21           c0.split[n] = -1;
22           memcpy(c1.split,split,sizeof(split));
23           c1.split[n] = 1;
24           PP::splitBy(n,c0,c1);
25       }
```

```
26      };
```

Для хранения информации о тех измерениях, по которым начальный гиперкуб уже был разрезан, мы храним массив *split*, содержащий значения $\{-1, +1, 0\}$. Последние три числа означают, что данный K-полугиперкуб отрезан по данному измерению как младшая/старшая половинки, либо еще не был «расположинен» по данному измерению вовсе.

Основная T-функция для параллельного вычисления интеграла выглядит следующим образом:

T-System

```
1       tfun double aInt(level_t l, GSCube c, const PtSet ps) {
2         double vol = c.volume(), average;
3         if (ps.variation(average) * vol < E)
4           return average * vol;
5         dim_t n = c.availSplit();
6         if (!(n < N)) {
7           if (l==1) {
8             return average * vol;
9           } else {
10            c.cubify();
11            n = 0; l--;
12          }
13        }
14        GSCube c0,c1; c.splitBy(n,c0,c1);
15        int k = ps.n / 2;
16        if (k < MP)
17          k = MP;
18        tval double r0 = ({ PtSet ps0(k,c0,&ps); aInt(l,c0,ps0);
19  });
20          tval double r1 = ({ PtSet ps1(k,c1,&ps); aInt(l,c1,ps1);
21  });
22          return (double)r0 + (double)r1;
23      }
```

Мы считаем G штук приближений к интегралу обычным методом Монте-Карло, оцениваем вариацию полученных значений, и, если она оказалась меньше порогового значения E, то полагаем в качестве конечного приближения *среднее арифметическое* от всех вычисленных G результатов.

Если вариация слишком велика, то мы вынуждены продолжить разбивать K-полугиперкуб по оставшимся (неиспользованным ранее) измерениям. Если таковых уже не осталось, то значит мы получили гиперкуб (вдвое меньшего диаметра) и можем начать процесс

с первого (по счету) измерения 0, уменьшив глубину оставшейся рекурсии l на единицу. Если же разбиение осуществимо, то мы вызываем функцию $aInt$ рекурсивно, пополнив необходимые множества новыми точками, и возвращаем сумму полученных значений. В этом месте происходит динамическое распараллеливание программы с точки зрения логики T-системы. Последняя автоматически переходит с T-вызовов на C-вызовы по достижению определенной глубины рекурсии, что легко заметить по общему количеству посчитанных T-функций в выдаваемой OpenTS статистической информации.

Константа MP определяет минимальное количество точек внутри каждого K-полугиперкуба и является одним из параметров, влияющих на точность вычислений.

Методика и результаты тестирования адаптивного алгоритма Для оценки точности и производительности данного метода была выбрана явно интегрируемая функция $f1$, представленная на приведенном ниже листинге одноименной C-функцией. В данном примере нами специально была выбрана функция с аналитически вычислимой первообразной, что позволяет явно получить точный ответ и далее использовать его для оценки погрешности исследуемого вычислительного метода.

График функции $f1$ имеет характерный одиночный максимум, и, чтобы усложнить ее поведение, просуммируем несколько экземпляров этой функции в разных точках, что даст нам результирующую функцию f, которую мы и будем интегрировать.

Для имитации вычислительной тяжести функции f была использована системная функция *usleep*, приостанавливающая вычислительный поток на заданное количество микросекунд:

T-System

```
1  double f1(double c[N], double a) {
2    double v = 1;
3    for (dim_t i=0; i<N; i++) {
4      double x = c[i] - a;
5      v *= pow(x*x+1,-1.5);
6    }
7    return v;
8  }
9  double f(double c[N]) {
10   usleep(F_T);
11   return f1(c,0) + f1(c,C);
12 }
```

Здесь C — константа, на которую мы сдвигаем график второго экземпляра функции $f1$ относительно первого.

Кратный неопределенный интеграл от нашей функции f может быть вычислен следующим образом (функция F):

T-System

```
double F1(double c[N], double a) {
  double v = 1;
  for (dim_t i=0; i<N; i++) {
    double x = c[i] - a;
    v *= x / sqrt(x*x+1);
  }
  return v;
}
double F(double c[N]) {
  return F1(c,0) + F1(c,C);
}
```

Соответственно, точное значение кратного определенного интеграла по заданному многомерному параллелепипеду определяется знакопеременной суммой кратной первообразной от функции f по всем его вершинам:

T-System

```
double exactInt(const PP& pp) {
  double s = 0;
  for (long m = (1L<<N)-1; m>=0; m--) {
    double c[N]; int sign=1;
    for (dim_t n=0; n < N; n++) {
      if (m & (1L<<n)) {
            c[n]   = pp[n].e;
          } else   {
            c[n]   = pp[n].b;
            sign   = -sign;
          }
        }
        s += sign * F(c);
    }
    return s;
}
```

Здесь мы для простоты полагаем, что размерность пространства $N < 63$, и обходим все вершины многомерного параллелепипеда путем интерпретации целых чисел, лежащих в диапазоне $[0..2 * *N - 1]$, как битовых масок для выбора начал и концов соответствующих ребер.

Параметры программы были выбраны так, чтобы точность вычислений интеграла была не хуже 1% для трехмерного куба. Такую точность обеспечивает стратифицированный метод Монте-Карло при делении каждого измерения на 64 части, что требует в итоге $64 * 64 * 64 = 262144$ вызова интегрируемой функции f.

Программа *производила* пять независимых расчетов с применением исследуемых алгоритмов, при этом печаталось полученное значение интеграла, погрешность относительно его точного значения, вычисленного аналитически, и требуемое количество вызовов функции f. Итоговое время выполнения программы измерялось в секундах.

Ниже приведены результаты запуска исследуемого алгоритма для трех разных режимов: без адаптивности, и с логикой адаптивности для одного и двух вычислительных потоков соответственно.

Неадаптивная версия (стратифицированный метод Монте-Карло):

Shell

```
$ time -p aint.t
[...]
Integral=12.711 calls=524287 f_calls=262144 error=0.5072%
Integral=12.779 calls=524287 f_calls=262144 error=0.030153%
Integral=12.738 calls=524287 f_calls=262144 error=0.29575%
Integral=12.724 calls=524287 f_calls=262144 error=0.40441%
Integral=12.74 calls=524287 f_calls=262144 error=0.27686%
Tasks activated: 40956
Taskboard visits: 41077
[...] real 243.69
```

Адаптивная версия, однопоточный режим:

Shell

```
$ time -p aint.t
[...]
Integral=12.744 calls=881 f_calls=41716 error=0.24514%
Integral=12.731 calls=941 f_calls=43116 error=0.34647%
Integral=12.698 calls=893 f_calls=41934 error=0.61011%
Integral=12.733 calls=919 f_calls=42490 error=0.33357%
Integral=12.707 calls=911 f_calls=42439 error=0.5392%
Tasks activated: 2194
Taskboard visits: 2315
[...] real 41.617
```

Адаптивная версия, многопоточный режим (два ядра CPU):

Shell

```
$ time -p aint.t -tct enableSMP
[...]
Integral=12.667 calls=897 f_calls=42037 error=0.85101%
Integral=12.742 calls=927 f_calls=42472 error=0.26569%
Integral=12.705 calls=907 f_calls=42561 error=0.55525%
Integral=12.776 calls=883 f_calls=41583 error=0.00057307%
Integral=12.757 calls=895 f_calls=41929 error=0.14655%
Tasks activated: 2174
[...]
Taskboard visits: 2616216
Idle time: 21.707
[...] real 31.463
```

В последнем запуске T-система зафиксировала значительный (около 11 секунд в пересчете с суммарного времени двух ядер) простой CPU, поскольку интегрируемая функция в данном эксперименте намеренно вызывалась для каждого K-полугиперкуба как C-функция (если вызывать ее как T-функцию, что актуально для реальных задач, то экземпляры ее вызовов равномерно распределяются по всем имеющимся вычислительным мощностям, и это совершенно не позволяет оценить, насколько хорошо распараллеливается *собственно логика адаптивного алгоритма*).

5. Выводы и направления для дальнейшей работы

Адаптивные алгоритмы всегда хорошо сочетались с концепцией динамического распараллеливания. Метастохастические методы, в которых между порождающим и порождаемым стохастическими процессами происходит содержательный анализ, естественно сочетаются с идеей адаптивности и перспективны прежде всего как эффективная комбинация интеллектуального и высокопараллельного счета. Авторы считают, что рассматриваемый в данной статье алгоритм является частным, простым представителем интересного семейства параллельных алгоритмов, базирующихся на следующих трех принципах:

- динамическое распараллеливание вычислений;
- интеллектуальный экспресс-анализ получаемых результатов;
- динамическая оптимизация параметров параллельного вычислительного алгоритма, в том числе и с учетом изменений, происходящих в вычислительной среде (отказы оборудования и т.д.).

Ключевым моментом здесь является возможность проведения **интеллектуального экспресс-анализа**. Обычно при разработке адаптивных алгоритмов ограничиваются относительно простыми критериями для реконфигурации вычислительного процесса. Это логично, так как любой проводимый в процессе счета анализ неминуемо тормозит вычисления.

Предлагаемый же подход в общем случае предполагает наличие нескольких уровней адаптивности: традиционный (с простыми критериями и правилами реконфигурации) для проведения локальной и интеллектуальный (со сложными критериями и правилами реконфигурации) для проведения глобальной оптимизации на мета-уровне, то есть на уровне алгоритма, динамически порождающего низкоуровневые алгоритмы с соответствующими вычислительными гранулами.

Такой подход может стать особенно плодотворным для получения эффективных адаптивных вычислительных методов для гибридных, сложно организованных суперЭВМ. Поэтому адаптация разработанного кода для гибридной вычислительной платформы T++/CUDA является приоритетной задачей, также как и тестирование на более сложных, представляющий практический интерес функциях. Исследования по этим направлениям планируется продолжить, а наиболее интересные результаты — изложить в последующих публикациях.

По результатам же экспериментальной программы для модельной задачи интегрирования априори-неизвестной функции можно сделать следующий вывод: для повышения параллелизма логики адаптивного алгоритма имеет смысл увеличить количество измерений, по которым разбиваются K-полугиперкубы. То есть на каждом шаге рекурсии переходить от K-полугиперкуба сразу к массиву из $2^m(K+m)$-полугиперкубов.

Впрочем, если количество вызовов функции f для каждой подобласти будет достаточно велико (либо сама функция f будет распараллелена при помощи конструкций T++ или вызовов MPI) OpenTS должна обеспечить традиционно высокий КПД, близкий к 100%, так как основной потенциал распараллеливания методов типа Монте-Карло заключен все-таки в независимости отдельных испытаний.

Список литературы

[1] В. А. Роганов, А. А. Кузнецов, Г. А. Матвеев, В. И. Осипов. «Реализация T-системы с открытой архитектурой для CUDA-устройств с поддержкой динамического параллелизма и для гибридных суперЭВМ на их основе», *Программные системы: теория и приложения*, **6**:1(24) (2015), с. 175–188, URL: http://psta.psiras.ru/read/psta2015_1_175-188.pdf ↑ 174

[2] В. А. Роганов, В. И. Осипов, Г. А. Матвеев. «Решение задачи Дирихле для уравнения Пуассона методом Гаусса–Зейделя на языке параллельного программирования T++», *Программные системы: теория и приложения*, **7**:3(30) (2016), с. 99–107, URL: http://psta.psiras.ru/read/psta2016_3_99-107.pdf ↑ 175

[3] В. А. Роганов, А. А. Кузнецов, Г. А. Матвеев, В. И. Осипов. «Адаптивный анализ надежности паролей при помощи гибридных суперЭВМ», *Программные системы: теория и приложения*, **6**:4(27) (2015), с. 139–155, URL: http://psta.psiras.ru/read/psta2015_4_139-155.pdf ↑ 175

[4] А. А. Кузнецов, В. А. Роганов, Г. А. Матвеев, В. И. Осипов. «Алгоритм динамического распараллеливания решения задачи адаптивного разбиения расчетной сетки для численного решения дифференциальных уравнений», *Программные системы: теория и приложения*, **6**:3(26) (2015), с. 53–60. ↑ 175

[5] https://ru.wikipedia.org/wiki/Метод_Монте-Карло. ↑

[6] С. М. Абрамов, В. А. Васенин, Е. Е. Мамчиц, В. А. Роганов, А. Ф. Слепухин. «Динамическое распараллеливание программ на базе параллельной редукции графов. Архитектура программного обеспечения новой версии T-системы», *Научная сессия МИФИ-2001*, Сборник научных трудов. Т. 2 (Москва, 22–26 января 2001 г.), с. 234. ↑

[7] Абрамов С. М., Кузнецов А. А., Роганов В. А.. «Кроссплатформенная версия T-системы с открытой архитектурой», *Вычислительные методы и программирование*, **8**:1(2) (2007), с. 175–180, URL: http://num-meth.srcc.msu.ru/zhurnal/tom_2007/v8r203.html ↑

[8] С. М. Абрамов, И. М. Загоровский, М. Р. Коваленко, Г. А. Матвеев, В. А. Роганов. «Миграция от MPI к платформе OpenTS: эксперимент с приложениями PovRay и ALCMD», *Международная конференция «Программные системы: теория и приложения»*. Т. 1 (Переславль-Залесский, октябрь 2006), Наука. Физматлит, М., 2006, с. 265–275. ↑

Рекомендовал к публикации *Программный комитет*
Пятого национального суперкомпьютерного форума ***НСКФ-2016***

Пример ссылки на эту публикацию:

С. М. Абрамов, В. А. Роганов, В. И. Осипов, Г. А. Матвеев. «Метастохастические адаптивные алгоритмы и их реализация в супервычислительной среде T++&MPI», *Программные системы: теория и приложения*, 2017, **8**:1(32), с. 173–191.

URL: http://psta.psiras.ru/read/psta2017_1_173-191.pdf

Об авторах:

Сергей Михайлович Абрамов

Чл.-корр. РАН, д.ф.-м.н. Директор ИПС им. А.К. Айламазяна РАН, ректор УГП им. А. К. Айламазяна. Научный руководитель от России суперкомпьютерных программ «СКИФ» и «СКИФ-ГРИД» Союзного государства. Область интересов: суперкомпьютерные, грид-и облачные технологии (аппаратура и программное обеспечение); компьютерные и сенсорные сети, IoT; метавычисления и их применение

e-mail: abram@botik.ru

Владимир Александрович Роганов

Научный сотрудник ИПС им. А.К. Айламазяна РАН. Разработчик современных версий T-системы, ведущий разработчик системы OpenTS. Принимал активное участие в суперкомпьютерных проектах Союзного государства России и Беларуси, в том числе в проектах «СКИФ» и «СКИФ-ГРИД»

e-mail: var@pereslavl.ru

Валерий Иванович Осипов

К.ф.-м.н., научный сотрудник ИПС им. А.К. Айламазяна РАН. Один из разработчиков системы OpenTS. Принимал участие в суперкомпьютерных проектах Союзного государства России и Беларуси

e-mail: val@pereslavl.ru

Герман Анатольевич Матвеев

Ведущий инженер-исследователь ИЦМС ИПС им. А.К. Айламазяна РАН. Один из разработчиков системы OpenTS. Принимал участие в суперкомпьютерных проектах Союзного государства России и Беларуси

e-mail: gera@prime.botik.ru

Sergey Abramov, Vladimir Roganov, Valerii Osipov, German Matveev. *Meta-Stochastic adaptive algorithms and theirs implementation with parallel programming system T++ & MPI.*

Abstract. The article describes parallel adaptive method for calculating multidimensional integrals. (*In Russian*).

Key words and phrases: dynamic parallelization, T-system with an open architecture, OpenTS, T++ programming language, adaptive algorithm, Meta-Stochastic process, scalable computing, multiparameter system, multidimensional integral, Monte Carlo method, recursive descent.

References

[1] V. A. Roganov, A. A. Kuznetsov, G. A. Matveyev, V. I. Osipov. "Implementation of T-System with an Open Architecture for Cuda Devices Supporting Dynamic Parallelism and for Hybrid Computing Clusters", *Programmnyye sistemy: teoriya i prilozheniya*, **6**:1(24) (2015), pp. 175–188 (in Russian), URL: http://psta.psiras.ru/read/psta2015_1_175-188.pdf

[2] V. A. Roganov, V. I. Osipov, G. A. Matveyev. "Solving the 2D Poisson PDE by Gauss-Seidel Method with Parallel Programming System OpenTS", *Programmnyye sistemy: teoriya i prilozheniya*, **7**:3(30) (2016), pp. 99–107 (in Russian), URL: http://psta.psiras.ru/read/psta2016_3_99-107.pdf

[3] V. A. Roganov, A. A. Kuznetsov, G. A. Matveyev, V. I. Osipov. "Adaptive Analysis of Passwords' Reliability using Computational Power of Hybrid Supercomputers", *Programmnyye sistemy: teoriya i prilozheniya*, **6**:4(27) (2015), pp. 139–155 (in Russian), URL: http://psta.psiras.ru/read/psta2015_4_139-155.pdf

[4] A. A. Kuznetsov, V. A. Roganov, G. A. Matveyev, V. I. Osipov. "Dynamic Parallel Algorithm of Mesh Adaptive Refinement for Numerical Solution of Differential Equations", *Programmnyye sistemy: teoriya i prilozheniya*, **6**:3(26) (2015), pp. 53–60 (in Russian).

[5] https://ru.wikipedia.org/wiki/Metod_Monte-Karlo (in Russian).

[6] S. M. Abramov, V. A. Vasenin, Ye. Ye. Mamchits, V. A. Roganov, A. F. Slepukhin. "Dynamic parallelization of programs based on parallel reduction of graphs. Software architecture of the new version of the T-system", *Nauchnaya sessiya MIFI-2001*, Sbornik nauchnykh trudov. V. 2 (Moskva, 22–26 yanvarya 2001 g.), pp. 234 (in Russian).

[7] Abramov S. M., Kuznetsov A. A., Roganov V. A.. "Cross-Platform Version of the T-System with an Open Architecture", *Vychislitel'nyye metody i programmirovaniye*, **8**:1(2) (2007), pp. 175–180 (in Russian), URL: http://num-meth.srcc.msu.ru/zhurnal/tom_2007/v8r203.html

[8] S. M. Abramov, I. M. Zagorovskiy, M. R. Kovalenko, G. A. Matveyev, V. A. Roganov. "Migrating from MPI to OpenTS Platform: experience with the PovRay and ALCMD applications", *Mezhdunarodnaya konferentsiya "Programmnyye sistemy: teoriya i prilozheniya"*. V. 1 (Pereslavl'-Zalesskiy, oktyabr' 2006), Nauka. Fizmatlit, M., 2006, pp. 265–275 (in Russian).

Sample citation of this publication:

Sergey Abramov, Vladimir Roganov, Valerii Osipov, German Matveev. “Meta-Stochastic adaptive algorithms and theirs implementation with parallel programming system T++ & MPI”, *Program systems: Theory and applications*, 2017, **8**:1(32), pp. 173–191. (*In Russian*).

URL: http://psta.psiras.ru/read/psta2017_1_173-191.pdf