Анализ производительности простой системы времени выполнения для акторного программирования на языке C++

С.В. Востокин[a], Е.Г. Скорюпина[a]

[a] *Самарский национальный исследовательский университет имени С.П. Королева, 443086, Московское шоссе, 34, Самара, Россия*

Аннотация

В работе представлена система времени выполнения Templet для акторного программирования высокопроизводительных вычислений на языке C++. Рассмотрен компактный код системы времени выполнения, использующий только стандартную библиотеку C++11, и показаны его отличия от классических реализаций модели акторов. Практическая значимость системы Templet доказывается на основе сравнительного исследования производительности трёх приложений: эталонного кода на языке C++, управляемого OpenMP; акторного кода на языке C++, управляемого системой Templet; акторного кода на языке Java, управляемого системой Akka. В качестве тестовой задачи рассмотрен численный алгоритм решения уравнения теплопроводности.

Ключевые слова: анализ производительности; промежуточное программное обеспечение, ориентированное на сообщения; акторный фреймворк, высокопроизводительные вычисления; язык C++

1. Введение

Предложенная Hewitt в 1973 году [1] концепция акторных вычислений в настоящее время не устарела и привлекает всё большее внимание разработчиков. Это обусловлено современной тенденцией широкого распространения аппаратных решений для массивно-параллельных вычислений. Одно из главных свойств модели акторов – возможность естественного описания неограниченного параллелизма. Поэтому активно развивающиеся технологии инфраструктурного ПО, интернета вещей и высокопроизводительных вычислений, использующие массивно-параллельные вычисления, хорошо укладываются в концепцию акторов [2].

В области инфраструктурного ПО и интернета вещей распространен фреймворк Akka [3] для интерпретируемых языков Scala и Java. Что касается высокопроизводительных вычислений, где доминируют компилируемые языки, акторы до настоящего времени использовались мало. Это, по нашему мнению, обусловлено сложившимся стереотипом о сложности реализации акторов для компилируемых языков. Новые свойства последних версий стандарта, начиная с C++11, привели к появлению эффективных и переносимых реализаций модели акторов и для компилируемого языка C++ [4].

Цель этого исследования – (1) предложить простую, являющуюся частью исходного кода приложения, реализацию акторов на языке C++11, (2) показать эффективность этой реализации в задачах высокопроизводительных вычислений.

Работа имеет следующую структуру. Вначале обсуждается тестовая задача для оценки производительности из области высокопроизводительных вычислений. Затем описывается реализация акторной системы времени выполнения Templet. Далее предложены три параллельных реализации тестовой задачи: на основе OpenMP, исследуемой системы Templet и фреймворка Akka. Описываются условия постановки вычислительного эксперимента и сравниваются результаты. Приводится заключение по работе.

2. Метод оценки эффективности: тест на основе уравнения теплопроводности

Для сравнительного тестирования мы использовали алгоритм, иллюстрирующий решение уравнения теплопроводности (листинг 1,2). Выбор обусловлен тем, что алгоритм описывает типовую реализацию часто используемого сеточного метода для одномерной декомпозиции области данных.

Константы W и H хранят ширину и высоту сеточной области field. Константа T – предельное число отсчетов по времени. Элементарная операция op (листинг 1) показывает применение дифференциального шаблона для пересчета значений поля на следующем временном шаге.

```
1 void op(int i)
2 {
3 for (int j=1; j<W-1; j++)
4   field[i][j]=(field[i][j-1]+field[i][j+1]+
5     field[i-1][j]+field[i+1][j])*0.25;
6 }
```

Листинг 1. Элементарная операция в тесте на основе уравнения теплопроводности.

Изменяя код элементарной операции op (листинг 1), можно получить алгоритмы решения других задач. Например, легко адаптировать алгоритм для трехмерной области field, не изменяя общую структуру управления счетом (листинг 2).

```
1 double seq_alg()
2 {
3 for (int t=1;t<=T;t++)
4 for (int i=1;i<H-1;i++) op(i);
5 }
```

Листинг 2. Последовательный алгоритм теста на основе уравнения теплопроводности.

Еще одной особенностью теста является повторное использование значений температурного поля field при расчете временного слоя по методу Зейделя. С алгоритмической точки зрения это вносит зависимость между итерациями по i (листинг 2) и делает нетривиальным распараллеливание кода на основе OpenMP.

3. Система времени выполнения Templet

Система акторных вычислений Templet состоит из трёх основных частей: двух примитивных операций (send и access) и функции рабочих потоков для обработки сообщений (tfunc). Далее рассматривается код для параллельного выполнения акторов в разделяемой памяти, использующий потоки стандартной библиотеки языка C++.

Код операции отправки сообщения send показан в листинге 3. Отправка заключается в помещении сообщения в общую для всей системы акторов очередь и извещении потока, который, возможно, ожидает появление сообщения в пустой очереди (строка 6). Очередь защищена мьютексом. Он захватывается в строке 5. Сообщение хранит ссылку на актора-адресата и признак передачи. Они устанавливаются в строке 4. В строке 3 записан страж повторной отправки. Повторная отправка сообщения – это аварийная ситуация, говорящая об ошибке в коде приложения.

```
1 inline void send(engine*e, message*m, actor*a)
2 {
3 if (m->sending) return;
4 m->sending = true;m->a = a;
5 std::unique_lock<std::mutex> lck(e->mtx);
6 e->ready.push(m); e->cv.notify_one();
7 }
```

Листинг 3. Примитивная операция «send».

Доступ к объекту-сообщению в контексте процедуры обработки актора разрешен, если функция *access* (листинг 4) вернет значение «истина». Другими словами, если (1) сообщение имеет ссылку на актора, который проверяет доступ; (2) сообщение не находится в доставке (строка 3). Это условие является инвариантным во время обработки сообщения в контексте конкретного актора, пока сообщение в вызове *access* в не будет отправлено при помощи операции *send*. Отправка сообщения разрешена при условии, что перед моментом отправки функция *access* (листинг 4) вернет значение «истина».

```
1 inline bool access(message*m, actor*a)
2 {
3 return m->a == a && !m->sending;
4 }
```

Листинг 4. Примитивная операция «access».

Код функции рабочего потока показан в листинге 5. Он реализует паттерн потокового пула [5]. Задачей в терминах данного паттерна является сообщение, находящееся на доставке. Такое сообщение имеет значение поля *sending==true*.

```
1 void tfunc(engine*e)
 2 {
 3  message*m; actor*a;
 4
 5for (;;){
 6  {
 7    std::unique_lock<std::mutex> lck(e->mtx);
 8    while (e->ready.empty()){
 9      e->active--;
10      if (!e->active){
11                        e->cv.notify_one(); return;
12                    }
13      e->cv.wait(lck);
14      e->active++;
15    }
16    m = e->ready.front();
17    e->ready.pop();
18  }
19  a = m->a;
20  {
21    std::unique_lock<std::mutex> lck(a->mtx);
22    m->sending = false;
23    a->recv(m, a);
24  }
25   }
26 }
```

Листинг 5. Функция рабочего потока и обращение к функции обратного вызова «recv».

Выполнив извлечение задачи из пула (строка 16), рабочий поток запускает процедуру обработки сообщения *recv* в акторе. Для запуска процедуры *recv*: (1) определяется актор-адресат сообщения (строка 19); (2) устанавливается блокировка на данном акторе (строка 21); (3) изменяется признак сообщения на *sending==false* (строка 22); активируется процедура *recv* для обработки сообщения (строка 23).

Заметим, что захваченные блокировки снимаются неявно при выходе потока из синтаксической области видимости объекта блокировки lck. Остановка вычислений в системе акторов происходит, когда нет активных рабочих потоков.

4. Параллельные алгоритмы для теста на основе уравнения теплопроводности

Мы реализовали три параллельных версии кода листинга 1, 2. Все версии основываются на следующих правилах распараллеливания. Начинать итерацию t по времени и i по пространству (t, i) можно, (1) если итерации (t-1, i+1) и (t, i-1) уже завершились; (2) либо, если t=1 и итерация (t, i-1) уже завершилась. Если итерация не имеет i+1 или i-1 соседней итерации, то считается, что эта отсутствующая итерация завершилась. Итерация (1,1) выполняется безусловно вначале вычисления. Алгоритм останавливается, когда для каждого пространственного отсчета по i выполнилось T итераций по t.

Рассмотренную логику вычислений можно реализовать на OpenMP, как показано в листинге 6. Идея распараллеливания состоит в том, что можно вычислять одновременного либо четные, либо нечетные итерации i на каждом отсчете t. Строгое выполнение правил вычислений обеспечивается дополнительными проверками в строках 5, 10, 15 листинга 6.

```
 1 void par_omp()
 2 {
 3 #pragma omp parallel shared(H,T)
 4 {
 5for (int t = 1; t <= (2 * T - 1) + (H - 3); t++){
 6
 7  if (t % 2 == 1){
 8 #pragma omp for schedule(dynamic,1)
 9    for (int i = 1; i < H - 1; i += 2)
10      if (i <= t && i > t - 2 * T) op(i);
11  }
12  if (t % 2 == 0){
13 #pragma omp for schedule(dynamic,1)
14    for (int i = 2; i < H - 1; i += 2)
15      if (i <= t && i > t - 2 * T) op(i);
16  }
17}
18 }
19 }
```

Листинг 6. Параллельный алгоритм для теста уравнения теплопроводности, использована технология OpenMP.

Акторные варианты алгоритма листинга 1,2 позволяют явно записать правила распараллеливания. Для этого каждому пространственному отсчету по i сопоставляется актор. В обоих акторных алгоритмах используется N=H-2 актора.

В реализации акторов на основе библиотеки Templet правила распараллеливания выражены строками 5-7 листинга 7. В строках 11, 12 актор уведомляет своих соседей i-1 и i+1 (если они есть) о завершении итерации (t,i) путем отправки сообщений.

```
 1 void recv(message* , actor* a)
 2 {
 3  int id = (int)(a - as);
 4
 5  if ((id == 0 || access(&ms[id - 1], a)) &&
 6  (id == N - 1 || access(&ms[id], a)) &&
 7  (ts[id] <= T)){
 8
 9  op(id+1); ts[id]++;
10
11  if (id != 0)     send(&e, &ms[id - 1], &as[id - 1]);
12  if (id != N - 1) send(&e, &ms[id], &as[id + 1]);
13 }
14 }
```

Листинг 7. Акторный параллельный алгоритм для теста уравнения теплопроводности, использована библиотека Templet.

В реализации акторов на основе библиотеки Akka правила распараллеливания выражены строками 7-9 листинга 8. В строках 13-20 актор уведомляет своих соседей i-1 и i+1 (если они есть) о завершении итерации (t,i) путем отправки сообщений. Заметим, что код обработки сообщений в листингах 7 и 8 для удобства сопоставления реализован идентично. Секция кода 22-24 требуется для остановки вычислений.

Оба акторных алгоритма имеют код инициализации акторов, который не приводится. Полный код тестовых примеров и акторной библиотеки Templet размещен по адресу https://github.com/Templet-language/newtemplet/tree/master/etc/comparison_with_openmp.

```
1 public void onReceive(Object message) {
2      if (((Integer) message) == id - 1)
3              access_ms_id_minus_1 = true;
4      if (((Integer) message) == id)
5              access_ms_id = true;
6
7      if ((id == 0 || access_ms_id_minus_1) &&
8              (id == N - 1 || access_ms_id) &&
9              (Main.time[id] <= Main.T)) {
10
11        Main.op(id + 1); Main.ts[id]++;
12
13        if (id != 0) {
14               Main.actors[id - 1].tell(id - 1, getSelf());
15               access_ms_id_minus_1 = false;
16        }
17        if (id != Main.N - 1) {
18               Main.actors[id + 1].tell(id, getSelf());
19               access_ms_id = false;
20        }
21     }
22     if (Main.time[id] == Main.T + 1 && id == Main.N - 1) {
23           Main.system.terminate();
24     }
24 }
```

Листинг 8. Акторный параллельный алгоритм для теста уравнения теплопроводности, использован фреймворк Akka.

5. Результаты

Вычислительные эксперименты проводились на компьютере с процессором Intel(R) Core(TM) i3-3220T RAM 4GB, Windows 10 x64. Программы на C++ компилировались в среде Microsoft Visual 2015. Для программы на Java использовались JDK версии 1.8 и библиотека Akka версии 2.4.17, развернутые на том же компьютере.

Сложность задачи задается параметром H. От H зависят другие два параметра пространственно-временной области расчета: W = H*2, T=H*2. Заметим, что H также определяет гранулярность вычислений. Чем больше H, тем крупнее последовательно обрабатываемые порции данных.

Столбцы данных таблицы 1 обозначают времена счета алгоритмов в секундах: T_1^{JAVA} – последовательного на языке Java; T_1^{NATIVE} – последовательного на языке C++; T_p^{AKKA} – параллельного на языке Java под управлением Akka; $T_p^{TEMPLET}$ – параллельного на языке C++ под управлением Templet; T_p^{OPENMP} – параллельного на языке C++, использующего OpenMP.

Для учета временных флуктуаций данные, представленные в таблице 1, прошли предварительную статистическую обработку. Каждый результат вычислений в таблице 1 основан на серии из 19 экспериментов. Результат содержит только значащие цифры, гарантирующие попадание в интервал [min,max] с фактором доверия 90% (min – минимальное, max – максимальное значение времени в серии из 19 экспериментов).

Таблица 1. Экспериментальное время вычислений тестов на основе уравнения теплопроводности

H	T_1^{JAVA} *(c)*	T_1^{NATIVE} *(c)*	T_p^{AKKA} *(c)*	$T_p^{TEMPLET}$, *(c)*	T_p^{OPENMP}, *(c)*
400	1.79	1.357	0.8	0.42	0.40
500	3.5	3.028	1.2	0.92	0.9
600	6.1	5.238	1.8	1.81	1.77
700	9.8	7.37	2.7	3.01	2.92
800	14.6	12.46	3.7	4.60	4.52
900	20.9	17.73	5.4	6.5	6.4
1000	28.7	21.09	7.6	9.0	8.9

Таблица 2. Относительная эффективность системы выполнения Templet:
$E_{AKKA} = T_p^{AKKA} / T_p^{TEMPLET}$ и $E_{OPENMP} = T_p^{OPENMP} / T_p^{TEMPLET}$

H	E_{AKKA} *(%)*	E_{OPENMP} *(%)*
400	190	95
500	130	99
600	99	98
700	90	97
800	80	98
900	83	98
1000	84	99

В таблице 2 представлена эффективность реализации теста с использованием предлагаемой системы времени выполнения относительно реализаций, использующих системы Akka и OpenMP. Значения E_{AKKA} и E_{OPENMP} показывают, какой процент составляет ускорение реализации Templet от ускорения эталонных реализаций на основе Akka и OpenMP.

Корректность распараллеливания проверялась путем поэлементной проверки на равенство значений температурного поля field, вычисленных последовательным и параллельным методом. Проводилось одинаковое для параллельного и последовательного метода случайное начальное заполнение field. Физическая интерпретация результатов расчета не проводилась, так как находится за рамками данного исследования.

6. Обсуждение

Эксперименты подтвердили высокую относительную эффективность предлагаемой простой реализации системы времени выполнения Templet для акторных вычислений. Система Templet имеет лишь незначительное отставание в производительности в выполненных тестах от OpenMP, а для малых размеров тестовой задачи H=400..600 не отстает Akka или даже превосходит ее.

Преимуществом акторных алгоритмов Templet и Akka является простота реализации и отладки. В них распараллеливание описывается с точки зрения простого поведения каждого отдельного актора. При использовании OpenMP требуется понимать глобальное состояние вычислений в каждый момент времени, что выражается в сложных граничных условиях циклов алгоритма листинга 6.

Наш алгоритм практически не уступает реализации OpenMP. Такой результат получен несмотря на то, что мы использовали выразительные возможности стандартной библиотеки C++11, чтобы упростить код, пренебрегая эффективностью. При необходимости возможна оптимизация на основе примитива compare-and-swap, как предложено в работе [2], и алгоритмов захвата работы (work stealing) [6].

Источником простоты нашей реализации акторов является отступление от классического подхода, предложенного Agha [7] и реализованного в известных акторных фреймворках и языках, например, Erlang [8], Scala [9], CAF [2] и других. Подход Agha предполагает, что сообщения – это некоторые *значения,* передаваемые между акторами. Они накапливаются в почтовом ящике – специальной системной структуре, ассоциированной с актором. Актор получает доступ к *значениям* сообщений.

В нашей реализации сообщения трактуются как *переменные*, хранящие значения. Программист не ограничен синтаксическими правилами в доступе к сообщению из любого актора в любой момент. Однако, доступ является осмысленным и не приводит к нарушениям логики только, если функция access() для пары сообщение-актор вернула значение *истина*. Такой подход не требует от библиотеки времени выполнения сложного кода для копирования значений между почтовым ящиком и фреймом вызова актора.

Хотя в известных реализациях используется синтаксическое ограничение доступа, его можно непреднамеренно нарушить, передавая в сообщениях ссылки вместо значений.

Тест также показал, что, несмотря на то, что нативная последовательная реализация теста превосходит реализацию на языке Java по производительности, параллельная реализация с использованием Akka оказывается лучшей для размерностей тестовой задачи от H=600 и более. Это можно объяснить более совершенным алгоритмом планирования у Akka, чем рассмотренный алгоритм планирования системы Templet, а также алгоритм планирования выбранной для тестирования реализации OpenMP.

7. Заключение

В работе предложена простая реализация акторной модели вычислений на языке C++11, и показана возможность ее применения в высокопроизводительных вычислениях. Тестовый пример решения уравнения теплопроводности иллюстрирует высокую эффективность рассмотренной реализации. Она приближается по эффективности к традиционной для этой задачи технологии OpenMP, в некоторых случаях превосходит Akka и позволяет снизить сложность кодирования.

Библиотека времени выполнения используется в предметно-ориентированном языке Templet [10] для реализации паттернов параллельных вычислений. Реализации паттернов применяются в решении задач нелинейной динамики при проектировании космических аппаратов [11].

Благодарности

Авторы выражают признательность РФФИ за частичную поддержку данных исследований в рамках гранта 15-08-05934 А.

Литература

[1] Hewitt, C. A universal modular ACTOR formalism for artificial intelligence // Proceedings of the 3rd IJCAI. SanFrancisco, CA, USA: Morgan Kaufmann Publishers Inc. – 1973. – P.235–45.

[2] Charousset, D., Revisiting Actor Programming in C++/ D. Charousset, R. Hiesgen, T.C. Schmidt // Computer Languages, Systems & Structures, 2016. – Vol. 56. – P. 105-131.

[3] Lightbend Inc. Akka. [Electronic resource]. — Access mode: http://akka.io (10.01.2017).
[4] Charousset, D. Native Actors – A Scalable Software Platform for Distributed Heterogeneous Environments/ C. Dominik, T.C. Schmidt, R. Hiesgen, M. Wählisch// Proceedings of the 4rd ACM SIGPLAN Conference on Systems Programming and Applications (SPLASH '13) Workshop AGERE!, New York, NY, USA:ACM, 2013.
[5] Schmidt, D.C. Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects. – John Wiley & Sons, 2013 – Vol. 2 – 700 p.
[6] Blumofe, R.D. Scheduling multithreaded computations by works tealing/ R.D. Blumofe, C.D. Leiserson // Proceedings of the 35th annual symposium on foundations of computer science (FOCS) – 1994, P.356–368.
[7] Agha, G. Towards a theory of actor computation / G. Agha, I.A. Mason, S. Smith, C. Talcott // Proceedings of CONCUR. Lecture notes on computer science. – Heidelberg: Springer-Verlag – 1992. – Vol.630. – P.565–579.
[8] Armstrong, J. Erlang – a survey of the language and its industrial applications // Proceedings of the symposium on industrial applications of Prolog (INAP96) – Hino,1996 – P.16–18.
[9] Haller, P., Odersky, M. Scala actors: unifying thread-based and event-based programming / P. Haller, M.Odersky // Theor Comput Sci – 2009. Vol. 410(23). – P. 202–220.
[10] Vostokin, S.V. Templet: a markup language for concurrent actor oriented programming // CEUR Workshop Proceedings – 2016. – Vol. 1638, P. 460-468.
[11] Doroshin, A.V. Heteroclinic Chaos and Its Local Suppression in Attitude Dynamics of an Asymmetrical Dual-Spin Spacecraft and Gyrostat-Satellites // The Part II – The heteroclinic chaos investigation, Communications in Nonlinear Science and Numerical Simulation – 2016. – Vol. 31 (1-3), P. 171-196.