ЭФФЕКТИВНОЕ ПРОГРАММИРОВАНИЕ УЗЛОВ МНОГОПРОЦЕССОРНОЙ ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ

С.Е. Киреев

Институт вычислительной математики и математической геофизики СО РАН

Введение

Известно, что реальная производительность современных вычислительных систем, реализующих численные алгоритмы, часто составляет не более 10% от пиковой. Одной из причин такого падения производительности вычислительных систем является низкая реальная производительность отдельного узла.

Целью исследования является выработка рекомендаций по написанию эффективных программ для узлов многопроцессорных вычислительных систем. Под эффективностью программы будем понимать время ее исполнения при реализации заданного алгоритма.

В качестве среды для проведения эксперимента использовались узлы двух вычислительных кластеров: МВС-1000М (ИВМиМГ, Новосибирск): SMP-узел – 2 процессора Alpha 21264 / 833 МГц [1] и T-Forge32 (Т-системы, Москва) [2]: SUMO-узел – 2 процессора Opteron 244 / 1800 МГц.

С учетом архитектуры узла эффективность достигается за счет параллельной работы всех исполнительных устройств. Параллельную работу могут обеспечить быстрый доступ к данным и отсутствие зависимостей по данным и управлению. На одном процессоре это достигается за счет:

1) учета организации подсистемы памяти,
2) использования оптимизирующего компилятора,
3) использования специализированных библиотек.

1. Учет иерархии памяти

Классическая структура иерархии памяти показана на рис. 1. Для достижения наибольшей скорости исполнения прежде всего необходимо, чтобы активный набор данных располагался как можно ближе к процессору [3].

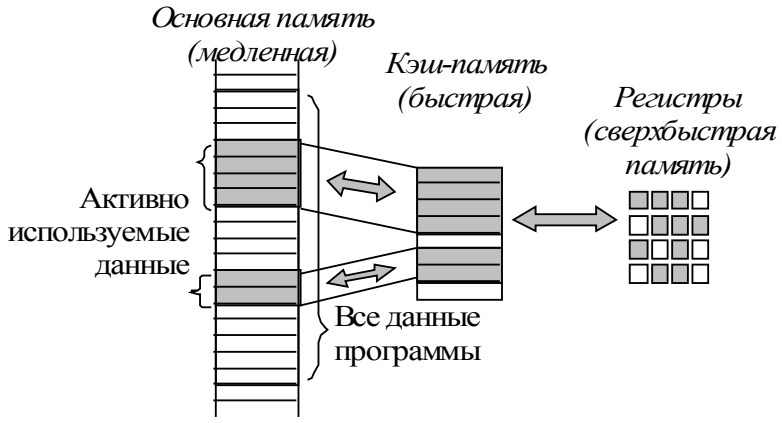


Рис. 1 Структура иерархии памяти

Рекомендация 1. Активный набор данных располагайте как можно выше по иерархии.

1.1. Использование регистров процессора

На самом верхнем уровне иерархии памяти расположены регистры процессора. Их число невелико, и они могут хранить только несколько значений, используемых процессором в данный момент.

Существуют стандарты, определяющие правила использования регистров процессоров в программах на различных операционных системах. Так, из общего числа регистров процессора часть может использоваться для хранения локальных переменных, часть – для хранения временных значений в процессе вычислений, часть – для передачи параметров в функции, часть – для возвращения результата и т.п. Зная число регистров, отводимых для хранения локальных переменных и параметров, мы определим максимальное число локальных переменных и число параметров подпрограмм, при которых программа будет работать наиболее быстро.

Рекомендация 2. Старайтесь не превышать указанные в табл. 1 количества локальных переменных и параметров подпрограмм.

Т а б л и ц а 1

Число регистров под локальные переменные и параметры подпрограмм

Процессор	Число регистров под локальные переменные		Число регистров под параметры подпрограмм	
	Целочисленные	Вещественные	Целочисленные	Вещественные
Alpha 21264	12	-	6	-
Opteron 244	15	16	6	8

1.2. Последовательный обход

Наиболее быстрым обходом данных является последовательный обход. Рассмотрим это на примере задачи перемножения двух квадратных матриц N×N. Если напрямую запрограммировать известную формулу $C_{ik} = \sum_{j=0}^{N-1} A_{ij} B_{jk}$, например, на языке Си, получим фрагмент программы, показанный в варианте 1 (рис. 2).

Вариант 1

```
for (i=0;i<N;i++)
 for (k=0;k<N;k++)
  for (j=0;j<N;j++)
   C[i][k]+=A[i][j]*B[j][k];
```

Вариант 2

```
for (i=0;i<N;i++)
 for (j=0;j<N;j++)
  for (k=0;k<N;k++)
   C[i][k]+=A[i][j]*B[j][k];
```

Рис.2 Варианты перемножения матриц

Зная, что массивы в языке Си хранятся по строкам, приходим к выводу, что элементы массива A перебираются последовательно, а элементы массива B – нет. Для ускорения работы программы нужно, чтобы элементы массивов перебирались последовательно. Для этого необходимо либо заранее транспонировать массив B, либо переставить циклы, как показано в варианте 2 (рис. 2).

Рекомендация 3. Старайтесь использовать последовательный обход данных (табл.2).

Таблица 2

Время перемножения матриц 1000×1000

Процессор	Вариант 1	Вариант 2	Ускорение
Alpha 21264	120,67 с	6,24 с	19,34
Opteron 244	16, 67 с	6,3 с	2,65

1.3. Учет ассоциативности кэш-памяти

Для вычислительных систем с ассоциативной кэш-памятью существует некоторый «плохой» шаг такой, что перебор данных в памяти с этим шагом выполняется намного дольше, чем с другими, отличающимися незначительно. Этот эффект называется «буксование» кэша.

Возникает он вследствие того, что для размещения элементов данных, расположенных в памяти с «плохим» шагом, кэш может предоставить только одну или несколько строк. Размер «плохого» шага вычисляется следующим образом: это числа, кратные размеру кэша (для ассоциативного), либо размеру кэша, деленного на число строк в множестве (для множественно-ассоциативного).

Рекомендация 4. При обходе данных избегайте идущих подряд обращений к элементам, расположенным с «плохим» шагом.

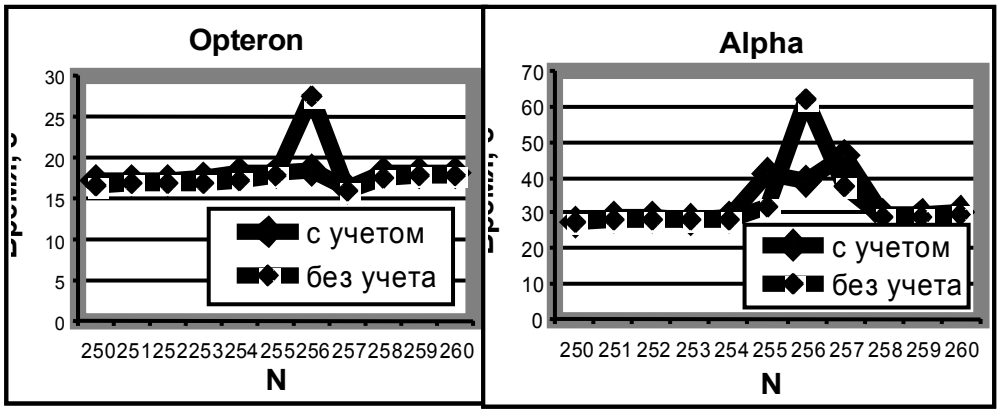


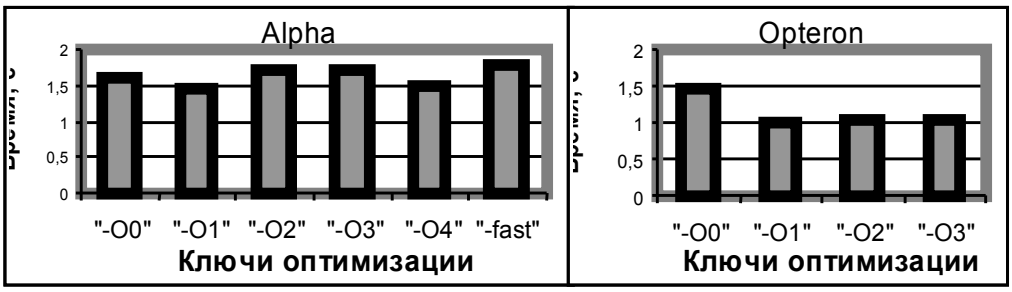
Рис.3 Результаты решения уравнения Пуассона методом Зейделя на сетке 256×256×N с учетом и без учета «плохого» шага

Эффект «буксования» кэша может возникнуть, например, при расчете элементов трехмерной сетки по семиточечной схеме (элемент и все соседи), если размер двумерного слоя равен «плохому» шагу (рис.3). Чтобы в таком случае избежать замедления, нужно изменить размер слоя, добавив после каждого слоя несколько неиспользуемых элементов с общей длиной не менее размера строки кэша.

2. Использование компилятора

Современные оптимизирующие компиляторы владеют большим числом техник оптимизации исполняемого кода, поэтому нельзя пренебрегать их возможностями при создании эффективных программ.

а) Вычисление числа Пи



б) Решение уравнения Пуассона методом Зейделя

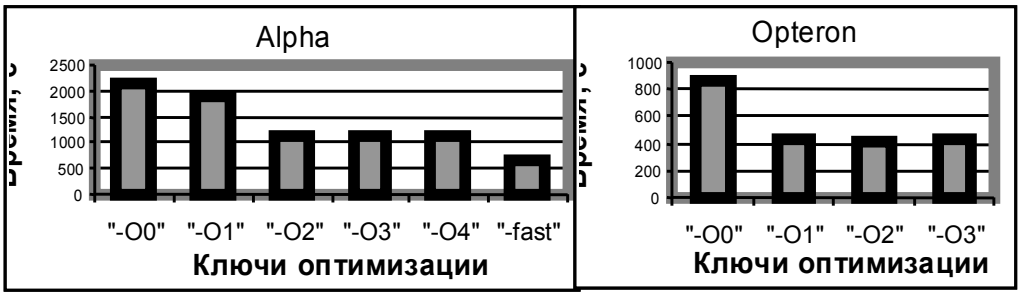


Рис. 4 Влияние оптимизационных ключей компилятора на время работы программ

С помощью опций компилятора можно включить использование той или иной техники или набора техник. Влияние техник оптимизации на эффективность получаемого кода во многом зависит от свойств конкретной программы (рис. 4).

Рекомендация 5. Для каждой конкретной программы выберите наилучший для нее набор ключей оптимизации.

3. Использование специализированных библиотек

Для многих предметных областей существуют библиотеки операций, оптимизированных под различные вычислительные системы. Они, как правило, реализованы очень эффективно, и их использование может сильно ускорить исполнение программы. Наиболее ярким примером таких библиотек являются BLAS и Lapack, которые содержат процедуры, реализующие многие операции линейной алгебры.

Рекомендация 6. По возможности используйте операции из эффективно реализованных специализированных библиотек (табл. 3).

Таблица 3

Время перемножения матриц 1000×1000

Процессор	Наилучший вариант, написанный вручную	Процедура из библиотеки BLAS	Ускорение
Alpha 21264	6,24 с	1,41 с	4,4
Opteron 244	6,3 с	0,64 с	9,8

Опыт пользователей показал, что использование данных рекомендаций позволяет в среднем на порядок увеличить производительность узла на задачах математической физики.

Литература

1. http://www2.sscc.ru/MVS-1000/MVS-1000.htm
2. http://www.t-systems.ru/
3. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. – СПб.: БХВ-Петербург, 2002. – 609 с.