УДК 681.3

ПРИМЕР ВИЗУАЛЬНОЙ МОДЕЛИ РАСПРЕДЕЛЕННОГО ВЫЧИСЛИТЕЛЬНОГО ПРОЦЕССА

© 2004 С.В. Востокин, С.А. Прохоров

Самарский государственный аэрокосмический университет

В статье рассматриваются этапы построения масштабируемой модели распределенного вычислительного процесса на визуальном языке моделирования. В качестве модельной задачи рассмотрен вычислительный процесс, встречающийся при численном решении уравнений математической физики на кластерных вычислительных системах.

Введение

Накопленный опыт применения высокопроизводительных компьютеров показывает, что при решении разных по своему содержанию задач используются похожие способы организации вычислений. В связи с этим в настоящее время актуальной является задача формализации типовых алгоритмических структур параллельных вычислительных процессов [1,2]. Для решения этой задачи удобно использовать визуальные методы моделирования. В статье рассматривается построение шаблона вычислительного процесса на визуальном языке моделирования, предложенном в работе [3].

Модель из двух процессов

Вычислительный процесс решения уравнений сеточными методами обычно сводится к многократному обходу матрицы чисел и выполнению некоторого преобразования элементов матрицы на каждом шаге. Типовым подходом к распараллеливанию таких алгоритмов является использование метода параллелизма данных. При этом матрица разбивается на параллельно обрабатываемые сегменты. Каждый сегмент принадлежит отдельному процессу. Граф взаимодействия процессов имеет вид цепочки, где каждый процесс (кроме двух крайних) взаимодействует с правым и левым соседом.

Рассмотрим вид вычислительного процесса, когда процедура преобразования элементов матрицы на каждом шаге обхода зависит как от значений, вычисленных на предыдущем шаге, так и от текущего значения.

В начале опишем шаблон, состоящий только из двух процессов. В этом случае цепочка взаимодействующих процессов будет выглядеть, как показано на рис. 1. Логика процессов **pLeft** и **pRight** показана на рис. 2 и рис. 3 соответственно. Структурные элементы модели, использованные для аннотирования диаграмм рис. 2 и рис.3, объявлены в файле модуля, код которого приводится ниже.

```
MODULE Laplace COMMENT
"Пример обобщенного"
"параллельного алгоритма"
;-------PROCESSes-------
```

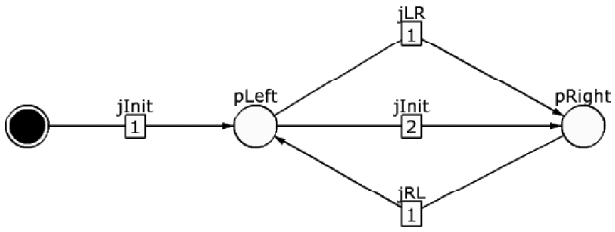

Рис. 1. Вид процесса **pMainPar1** модуля **Laplace**

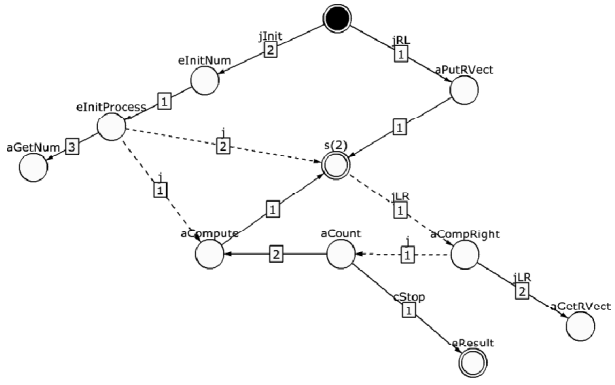

Рис. 2. Вид процесса **pLeft** модуля **Laplace**

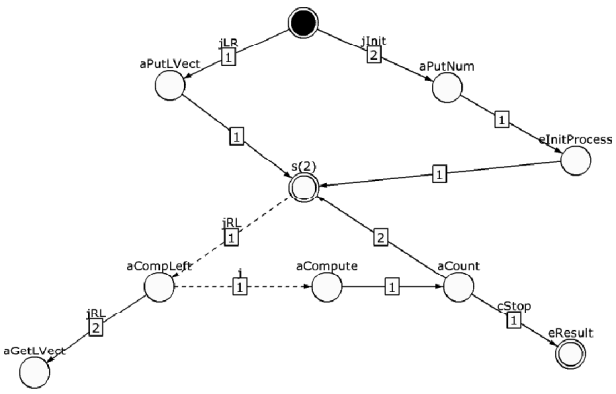

Рис. 3. Вид процесса `pRight` модуля `Laplace`

```
  PROCESS pMainPar1
  PROCESS pLeft
  PROCESS pRight
;-------JOBs-------
  JOB j COMMENT "работа процесса"
  JOB jInit COMMENT
    "инициализация программы"
  JOB jRL COMMENT
    "передача данных из правого"
    "в левый сегмент"
  JOB jLR COMMENT
    "передача данных из левого"
    "в правый сегмент"
;-------TYPEs-------
  TYPE tVECT COMMENT "вектор  double"
  TYPE tSEG  COMMENT "матрица double"
  TYPE tINT  COMMENT "целые   int"
;-------VARiables-------
  VAR vNum  TYPE tINT COMMENT
    "номер сегмента"
  VAR vCount TYPE tINT COMMENT
    "текущее число"
    "итераций до остановки"
  VAR vLVect TYPE tVECT COMMENT
    "массив 'левый вектор'"
  VAR vSeg   TYPE tSEG COMMENT
    "массив 'сегмент'"
  VAR vRVect TYPE tVECT COMMENT
    "массив 'правый вектор'"
;------PORTs------
  PORT eInitNum COMMENT
    "запись начального"
    "значения номера сегмента"
    INPUT  vNum
  END
  PORT eInitProcess COMMENT
    "инициализация процесса"
    OUTPUT vNum
    INPUT vCount,vSeg,vRVect
  END
  PORT eResult COMMENT
    "вывод результата вычислений"
    OUTPUT vNum,vSeg
  END
;------ACTIONs------
  ACTION s COMMENT "заглушка" END
  ACTION aPutNum COMMENT
    "инициализация номера сегмента"
    PUT vNum
  END
  ACTION aGetNum COMMENT
    "вычисление номера сегмента и"
    "подготовка его к передаче"
    PARAM vNum;vNum=vNum+1
    GET vNum
  END
  ACTION aCompute COMMENT
    "пересчет сегмента"
    PARAM vSeg
  END
  ACTION aCompLeft COMMENT
    "пересчет левой границы сегмента"
    PARAM vSeg,vLVect
  END
  ACTION aCompRight COMMENT
    "пересчет правой"
    " границы сегмента"
    PARAM vSeg,vRVect
  END
  ACTION aCount COMMENT
  "уменьшение счетчика итераций на 1"
    PARAM vCount; vCount=vCount-1
  END
  ACTION aGetLVect COMMENT
    "подготовка 'левого вектора'"
    "к передаче"
    GET vLVect
  END
  ACTION aGetRVect COMMENT
    "подготовка 'правого вектора'"
    "к передаче"
    GET vRVect
  END
  ACTION aPutRVect COMMENT
  "выгрузка вектора в процесс справа"
    PUT vLVect TO vRVect
  END
  ACTION aPutLVect COMMENT
   "выгрузка вектора в процесс слева"
  PUT vRVect TO vLVect
  END
;-------CONDITIONs-------
  CONDITION cStop COMMENT
    "проверка условия vCount==0"
    PARAM vCount
  END
END
```

Диаграммы процессов рис.2 и рис.3 читаются следующим образом. Вершины графов обозначают обобщенные последовательные процедуры, из которых состоит вычислительный процесс. Дуги задают порядок

активации процедур внутри каждого процесса. Сплошная дуга означает активацию непосредственно по завершении процедуры при истинности условия, помечающего дугу. Пунктирная дуга означает планирование новой работы (именованной цепочки активаций процедур) по завершении в процессе текущей активной работы. Работы именованы и имеют внутреннее состояние. Имя работе присваивается при ее планировании (помечая пунктирную дугу именем работы). По имени работы можно осуществлять выбор следующей запускаемой процедуры (помечая сплошную дугу именем работы) и передавать значения из процесса в процесс с использованием внутреннего состояния работы (используя команды **GET** и **PUT**). Назначение конкретных структурных элементов модели описано в комментариях к файлу модуля, текст которого приведен выше.

Переход к масштабируемой модели

Для описания реальных, используемых на практике, процессов нужно иметь возможность масштабирования модели. То есть, модель должна легко настраиваться на произвольное требуемое число процессов. Для кластерных вычислительных систем это число составляет 10-20 процессов. Массово параллельные суперкомпьютеры требуют наличия порядка тысячи процессов, а GRID системы — порядка миллиона процессов. Предложенный способ моделирования позволяет, не увеличивая структурную сложность модели, масштабировать ее на заданное число процессов.

Масштабирование нашей модели заключается в разделении области вычислений больше, чем на два сегмента. При таком разделении получаются сегменты, имеющие как правую, так и левую вычисляемую границу. Для вычисления такого сегмента требуется определить новый процесс **pCenter**, показанный на рис.4. Процесс описывается при помощи ранее определенных в файле модуля структурных элементов. Он представляет собой комбинацию процессов **pLeft** и **pRight**. Ядром процесса является цикл пересчета сегмента, состоящий из действий **aCompLeft**, **aCompute** и **aCompRight**. Синхронизация работы этого цикла с другими процессами и инициализация организуется аналогично процессам **pLeft** и **pRight**.

При использовании процесса **pCenter** в нашей модели диаграмма процесса **pMainPar2**, координирующего вычисления, будет выглядеть, как показано на рис. 5.

Легко заметить, что, добавляя в цепочку рис. 5 произвольное число процессов типа **pCenter**, можно добиться масштабируемости модели. Однако проблема заключается в том, что при таком способе масштабирования линейно увеличивается структурная сложность модели. Он может быть использован для простых кластерных систем, но для массово-параллельных систем уже не подходит.

Выход заключается в построении дерева управляющих процессов при помощи модулей с параметрами. При этом можно добиться эффекта экспоненциального роста сложности описываемой моделью системы при незначительном линейном росте структурной сложности самой модели.

Определим модуль с параметрами **Template**, содержащий объявление процесса **pCenter**:

```
MODULE Template PARAM M, Main
COMMENT "шаблон процесса"
        "в центральном сегменте"
PROCESS pCenter
END
```

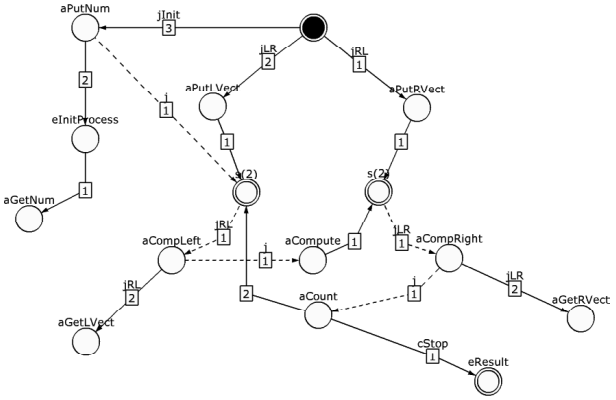

Рис. 4. Вид процесса **pCenter** модуля **Laplace**

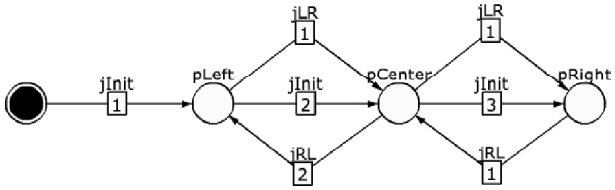

Рис. 5. Вид процесса **pMainPar2** модуля **Laplace**

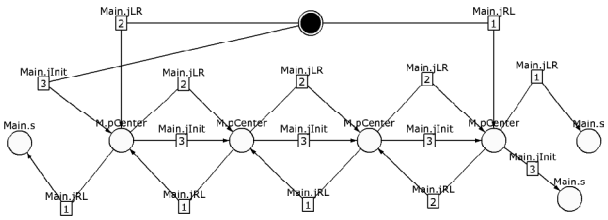

Рис. 6. Вид процесса **pCenter** модуля **Template**

Диаграмма процесса **pCenter** модуля **Template** показана на рис. 6.

Процесс **pCenter** модуля **Template** — это рекурсивный процесс, в каждой вершине **M.pCenter** которого (при связывании модулей) будет находиться либо процесс **pCenter** модуля **Template**, либо, при завершении рекурсии, процесс **pCenter** из модуля **Laplace**, показанный на рис. 4. Для организации описанной рекурсии потребуется объявить следующие элементы модуля **Laplace** экспортируемыми:

```
JOB jInit EXPORT
JOB jRL   EXPORT
JOB jLR   EXPORT
ACTION s  EXPORT END
```

Главным управляющим процессом модуля **Laplace** будет процесс **pMainPar3**, показанный на рис. 7. Кроме этого, в модуль **Laplace** нужно добавить соответствующие команды импорта модулей для связывания с модулем **Template**.

Команды импорта выглядят следующим образом:

```
IMPORT Laplace  AS Main
IMPORT Template AS Center4
       PARAM Main,Main
IMPORT Template AS Center16
       PARAM Center4,Main
IMPORT Template AS M
       PARAM Center16,Main; Center64
```

Приведенные выше команды добавляют 64 центральных процесса типа **pCenter** в цепочку процессов. Каждое новое добавленное предложение импорта увеличивает число процессов **pCenter** в цепочке в 4 раза (по числу процессов **M.pCenter** в процессе **pCenter** модуля **Template**).

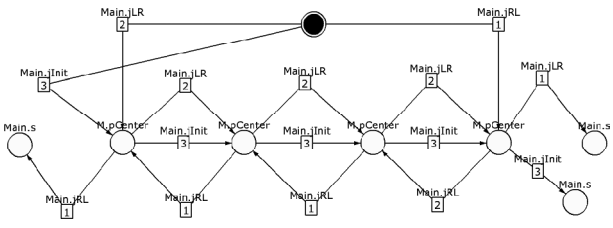

Рис. 7. Вид процесса **pMainPar3** модуля **Laplace**

Рассмотрим, за счет чего это происходит. Параметризованный модуль имеет несвязанные параметры в своем заголовке. Такой модуль является абстрактным, он обозначает не один конкретный модуль, а множество модулей. Связывая параметры модуля подстановкой конкретных модулей (модулей без параметров, или со связанными параметрами), мы получаем новые экземпляры конкретных модулей для данного параметризованного модуля. Обозначим структурный элемент модели как *<имя_модуля>***(***<список_параметров>***).** *<имя_объекта>*. Определим значения параметров, задаваемые предложениями импорта.

```
Main:=Laplace
Center4:=Template(Laplace,Laplace)
Center16:=Template(Template(Laplace,
          Laplace),Laplace)
M:=Template(Template(Template(
    Laplace,Laplace),Laplace),Laplace)
```

Видно, что в нашей модели абстрактный параметризованный модуль **Template** порождает три разных конкретных модуля. В процессе **pMainPar3** рис. 7 ссылка **M.pCenter** обозначает процесс с полным именем **Template(Template(Template (Laplace,Laplace),Laplace), Laplace).pCenter**. Этот процесс, в свою очередь, построен из процессов с полным именем **Template(Template(Laplace, Laplace,Laplace).pCenter.** На следующем уровне по дереву процессов находятся процессы с полными именами **Template(Laplace,Laplace).pCenter**, которые, наконец, состоят из процессов **Laplace.pCenter**.

Заключение

Нами построена типовая схема параллельного вычислительного процесса, встречающегося при численном решении уравнений в частных производных. Для задания схемы применен визуальный язык моделирования. Число процессов в использованном методе всегда является заданным, но его можно сделать достаточно большим для адекватного описания вычислительных про-

цессов в современных параллельных компьютерах кластерной архитектуры, ориентированных на взаимодействие путем обмена сообщениями.

СПИСОК ЛИТЕРАТУРЫ

1. *Берзигияров П.К.* Программирование на типовых алгоритмических структурах с массивным параллелизмом // Вычислительные методы и программирование. 2001. Т. 2.
2. *MacDonald S., Szafron D., Schaeffer J., Bromling S.* Generating parallel programs frameworks from parallel design patterns // Proc. Of EuroPar'2000. Berlin: Springer-Verlag, 2000.
3. *Востокин С.В.* Метод описания пространственно распределенных параллельных процессов. Самара: СНЦ РАН. 2004.

VISUAL MODEL SAMPLE OF A DISTRIBUTED COMPUTATIONAL PROCESS

© 2004 S.V. Vostokin, S.A. Prohorov

Samara State Aerospace University

The article describes development stages of a scalable model of the distributed computational process using visual modeling language. A sample computational process implemented on cluster computer architecture from the field of applied physics discussed in detail.