© 2012 г. С.В. ВОСТОКИН, д-р техн. наук
В.Г. ЛИТВИНОВ
Д.Д. МАКАГОНОВА
А.Р. ХАЙРУТДИНОВ
(Самарский государственный аэрокосмический университет им. академика)
С.П. Королева (национальный исследовательский университет), г. Самара

ВИЗУАЛЬНОЕ МОДЕЛИРОВАНИЕ ПАРАЛЛЕЛЬНЫХ АЛГОРИТМОВ В ПРОЦЕССНО-ОРИЕНТИРОВАННОЙ НОТАЦИИ TEMPLET

Представлен метод моделирования систем с внутренним параллелизмом на основе процессного подхода. Рассмотрены: декомпозиция процессов на процедуры обработки сообщений; метод описания протоколов взаимодействия процессов; графическая нотация для визуализации модели процессов; динамика модели. В качестве иллюстрации приведены примеры параллельных алгоритмов метода переменных направлений, вычисления с портфелем задач, конвейера.

VISUAL MODELING OF PARALLEL ALGORITHMS IN PROCESS-ORIENTED NOTATION TEMPLET / S.V. Vostokin, V.G. Litvinov, D.D. Makagonova, A.R. Khairutdinov (Samara State Aerospace University, 34 Moskovskoe shosse, Samara 443086, Russia, E-mail: easts@mail.ru). The method for modeling systems with internal parallelism based on processes-oriented approach is proposed. We discuss decomposition of processes on procedures for message handling, specification of process interaction protocols, graphical notation for visualization of the process model, model dynamics. Discussion is illustrated by examples of parallel algorithms for the altering directions method, the bag-of-tasks method, and the pipeline method.

1. Введение

Процессно-ориентированный метод представления параллелизма применяется в задачах численного моделирования, в системах сбора и обработки данных, при описании организационных и производственных процессов [1-4]. Используемые нотации, в основном, сходны между собой на верхнем уровне процессов и их взаимосвязей. Однако, применяя специальные приемы описания логики работы самих процессов и протоколов их взаимодействия, можно получить более адекватное и удобное представление параллелизма в моделируемой предметной области. В работе рассматривается метод моделирования «Templet», предназначенный для описания параллельных численных алгоритмов [5]. Метод является усовершенствованным вариантом графической нотации системы автоматизации параллельного и распределенного программирования Graphplus templet [6], разрабатываемой в рамках исследовательского проекта «Граф Плюс»

(graphplus.ssau.ru) Самарского государственного аэрокосмического университета (национального исследовательского университета).

Технические средства организации параллельных вычислений в задачах численного моделирования доступны и разнообразны: многоядерные настольные системы; кластеры и суперкомпьютеры; распределенные вычисления в сети интернет; GPGPU-процессоры. Однако существует проблема различия компетенций прикладного программиста и системного программиста. Прикладной программист знает, как по аналитической модели разработать численный метод и построить последовательную программу. В то же время понимание аппаратных особенностей, низкоуровневых системных средств, операционных систем, различных API параллельного программирования, методов синхронизации процессов является компетенцией системного программиста. Поэтому актуальна организация взаимодействия специалистов для эффективного решения задач моделирования. Целью работы является демонстрация подхода, основанного на графическом языке моделирования, который может являться основой такого взаимодействия. В процессе разработки графической нотации Templet решались следующие задачи:

1) построение модели с процедурной семантикой;
2) обеспечение минимального числа графических примитивов и простой семантики исполнения;
3) понятность для прикладного программиста правил преобразования модели в код;
4) совместимость с современными технологиями программирования.

Основа языка Templet - модель процессов диффузного типа. В данной модели конструктивными элементами являются пассивные процессы и активные двунаправленные каналы, соединяющие их. Логика процессов, логика каналов, а также способ композиции каналов и процессов представляются тремя типами диаграмм. Модель позволяет выполнить декомпозицию параллельного алгоритма на последовательные процедуры и типы данных, передающихся в сообщениях. При этом строго определяется контекст вызова процедур в терминах определенных пользователем типов данных для сообщений и состояний процессов.

В первой части работы дается полное описание графической нотации и семантики исполнения на примере процесса типа «Разветвление-Слияние». Далее рассматриваются некоторые прикладные параллельные алгоритмы.

2. Алгоритм «Разветвление-Слияние». Описание графической нотации

Параллельная система в целом моделируется как композиция элементов модели «процесс» и элементов модели «канал». Интерфейсными элементами процессов, участвующих в композиции, являются порты. Система «разветвление-слияние» ForkJoin в описываемой нотации представлена на рис. 1. Родительский процесс p выдает задания дочерним процессам c1 и c2, а затем выполняет обработку их ответов.

Пометки графа на рис. 1 обозначают: p:Parent – процесс p типа Parent; p1->p:Call – клиентский порт p1 (процесса p) связан с серверным портом p (процесса c1) каналом, имеющим тип Call. Точка на дуге указывает положение серверного порта.

Каналы. Диаграмма канала описывает объекты, задающие информационные связи между процессами. Диаграмма канала показывает, какие именно сообщения передаются между процессами, и определяет возможный порядок их передачи. Во взаимодействии по каналу участвуют два процесса: клиент и сервер. Возможна передача произвольного

числа сообщений в обе стороны. В любой момент времени передается не более одного сообщения.

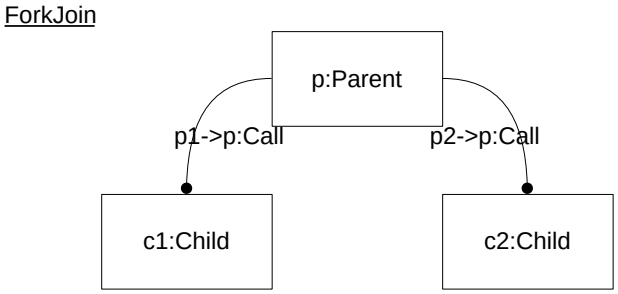

Рис. 1. Композиция процессов в системе «разветвление-слияние».

На рис. 2 показан протокол взаимодействия между родительским и дочерним процессами в системе «разветвление-слияние», описанный каналом Call. Пометки графа на рис. 2 обозначают: s0 – начальная вершина процесса-клиента; s1 – вершина процесса-сервера; s2 – вершина процесса-клиента; call и ret – передаваемые сообщения.

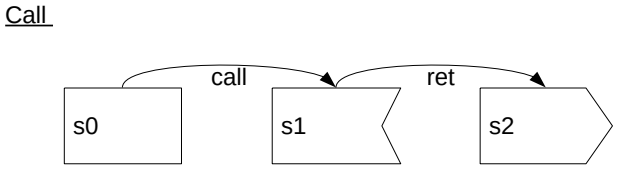

Рис. 2. Протокол взаимодействия процессов в системе «разветвление-слияние».

Состояние канала – это переменная, хранящая дугу с передаваемым сообщением и признак доставки сообщения. В момент отправки признак имеет значение «отправлено». Рано или поздно признак примет значение «доставлено», начнется обработка данного сообщения, и будет возможен его прием. В начальном состоянии переменная хранит специальное значение, сигнализирующее об отсутствии сообщения.

В начальном состоянии возможна передача сообщений процессом-клиентом по дугам, исходящим из начальной вершины (с пометкой как у s0 на рис. 2). Новым состоянием канала будет дуга, соответствующая переданному сообщению.

Если состоянием канала является дуга, то возможен прием сообщения, помечающего дугу. Принять сообщение может процесс-клиент, если вершина, в которую входит дуга, вершина процесса-клиента. Иначе – сообщение может принять процесс-сервер. Принимающий сообщение процесс может отправить ответное сообщение из тех, которые помечают дуги, исходящие из рассматриваемой вершины. Аналогично, новым состоянием канала будет дуга, соответствующая переданному сообщению.

Граф на рис. 2 показывает, что вначале клиент (родительский процесс p на рис. 1) передает дочернему процессу сообщение call, затем получает от него сообщение ret. На этом взаимодействие заканчивается.

Процессы. Диаграмма процесса описывает алгоритм обработки поступающих сообщений. Результатом обработки может являться отправка новых сообщений и/или изменение состояния процесса. Процессы являются пассивными, управляемыми сообщениями: алгоритм обработки запускается только при поступлении сообщения. Алгоритм обработки не может быть прерван другим сообщением до его завершения.

Примеры графических обозначений, используемых на диаграммах процессов, показаны на рис. 3 и рис. 4.

Пометки графов обозначают: fork – начальная вершина, запускающая алгоритм обработки в начальном состоянии; call и ret – получаемые или отправляемые сообщения;

p1 и p2 – клиентские порты процессов; p – серверный порт процесса; join, proc, error – методы обработки сообщений в процессах; yes, no – передача управления в случае успешного или неуспешного завершения процедуры обработки.

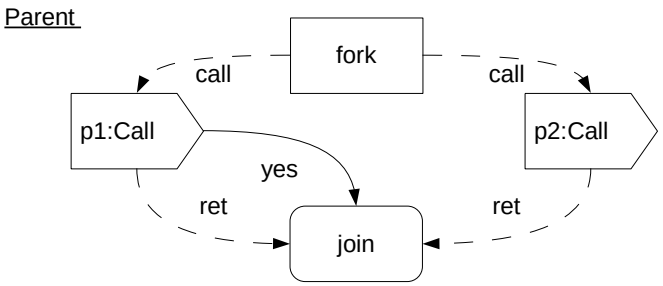

Рис. 3. Родительский процесс в системе «разветвление-слияние».

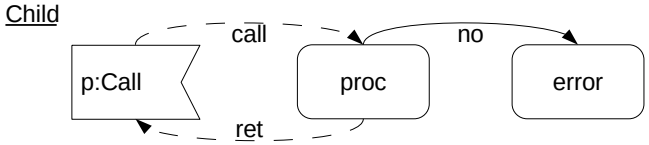

Рис. 4. Дочерний процесс в системе «разветвление-слияние».

Состояние процесса – это переменная, хранящая значение текущей вершины графа процесса. Когда обработка не выполняется, переменная хранит специальный признак останова. Если имеется сообщение, отправленное процессу, то рано или поздно начнется обработка данного сообщения с вершины-порта (p, p1 или p2 на рис. 3, рис. 4). Переменная состояния процесса примет значение соответствующей вершины, и будет возможен прием сообщения.

Правило передачи управления для портов:

1) если имеется исходящая дуга с пометкой yes, то управление передается по ней;
2) если имеется исходящая дуга, помеченная поступившим сообщением, то управление передается по данной дуге.

Правило запуска процедуры обработки сообщений. Процедура запускается, если одновременно:

1) переменная состояния принимает значение вершины, помеченной данной процедурой;
2) возможен прием сообщений, ведущих в эту вершину;
3) возможна отправка сообщений, исходящих из данной вершины.

Отправка исходящих сообщений выполняется, если одновременно:

1) процедура была запущена;
2) процедура вернула признак успешного завершения.

Правило передачи управления для процедур:

1) передача управления по дуге с пометкой yes происходит, если процедура была запущена и вернула признак успешного завершения;
2) иначе происходит передача управления по дуге с пометкой no.

Если нет исходящих из вершин дуг, по которым можно выполнить переход в текущей ситуации, переменная состояния процесса принимает значение признака останова.

Рассмотрим примеры процессов рис. 3 и рис. 4. Вычисления начинаются с выполнения метода fork процесса Parent. Он формирует и отправляет сообщения call дочерним процессам по портам p1 и p2. Так как из вершины не исходят дуги с пометками yes или

no, на этом обработка заканчивается. Сообщения call попадают в связанные дочерние процессы Child через порт p. Запускается метод proc. Его параметрами являются поступившее сообщение call и формируемое для отправки сообщение ret. В случае неуспешного завершения метода proc отправка сообщения ret не производится, а запускается метод error. Иначе в родительский процесс отправляется сообщение ret. При поступлении сообщения на порт p1 в процесс Parent управление немедленно передается в метод join. Альтернативную возможность передачи управления реализует порт p2. Здесь происходит проверка типа поступившего сообщения. Метод join родительского процесса Parent запускается, когда ответные сообщения ret придут от обоих дочерних процессов.

Отметим некоторые особенности представленной нотации. Диаграммы процессов внешне напоминают блок-схемы алгоритмов. Однако в приведенной нотации имеется несколько типов начальных вершин (fork и p1 на рис. 3, p на рис. 4). Не делается различий между процедурой и принятием решения, что также используется в некоторых нотациях [4]. Поток управления может пройти через вершину-метод, при этом метод может не выполниться, если не соблюдены некоторые условия. Данные свойства позволяют удобно описывать выбор метода обработки поступающих в процесс сообщений.

В отличие от блок-схем точно задан контекст запуска метода. Определено, какие сообщения пересылаются между процессами и их порядок. Это позволяет провести анализ непротиворечивости графического описания алгоритма.

Семантика исполнения модели по построению гарантирует отсутствие состояния состязания (race condition) при реализации алгоритма в общей памяти и состояния тупика (deadlock). Хотя для диффузных моделей возможна остановка в незавершенном состоянии, данная ситуация более проста, чем состояние тупика.

3. Примеры параллельных алгоритмов

3.1. Метод переменных направлений с функциональной декомпозицией

Рассмотрим, каким образом может быть осуществлено распараллеливание численных моделей, основанных на методе переменных направлений. Пример также иллюстрирует технологию разработки произвольных последовательно-параллельных программ в нотации Templet.

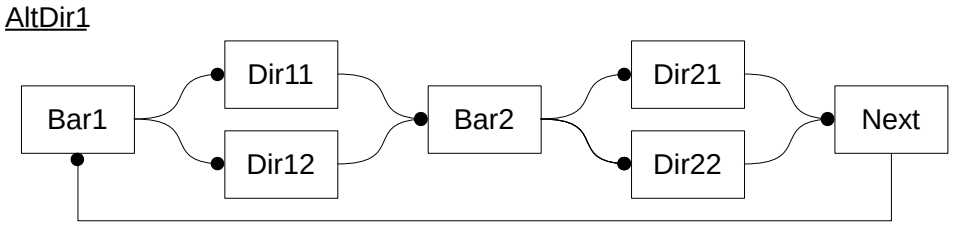

Рис. 5. Композиция процессов в методе переменных направлений с функциональной декомпозицией.

Структура процессов для функциональной декомпозиции показана на рис. 5. Процессы слева Dir11 и Dir12 выполняют вычисления горизонтальных независимых прогонок на ½ шага, процессы справа Dir21 и Dir22 выполняют вычисления вертикальных независимых прогонок при окончании шага (или наоборот: левые процессы – вертикальные, правые – горизонтальные прогонки). Реализация барьера по окончании ½ шага изображена на рис. 6 (б). Метод bar отправляет сообщения, если закончились все входящие параллельные ветви. Контроль корректности последовательно-параллельной

структуры реализуется при помощи ответных сообщений ret в каналах Link на рис. 6 (а). В программной реализации по модели данные сообщения не обязательны. Другие два барьерных процесса устроены аналогично процессу рис. 6 (б).

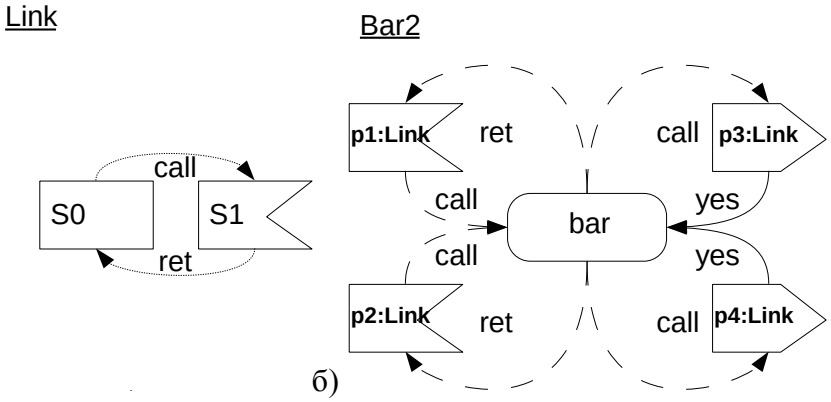

Рис. 6. Описание протокола канала (а) и барьера (б) в методе переменных направлений с функциональной декомпозицией.

3.2. Метод переменных направлений с декомпозицией по данным

Алгоритм переменных направлений также можно распараллелить в технике декомпозиции по данным. Ниже на рис. 7 и рис. 8 представлен пример построения параллельных алгоритмов для двумерных сеток из процессорных элементов.

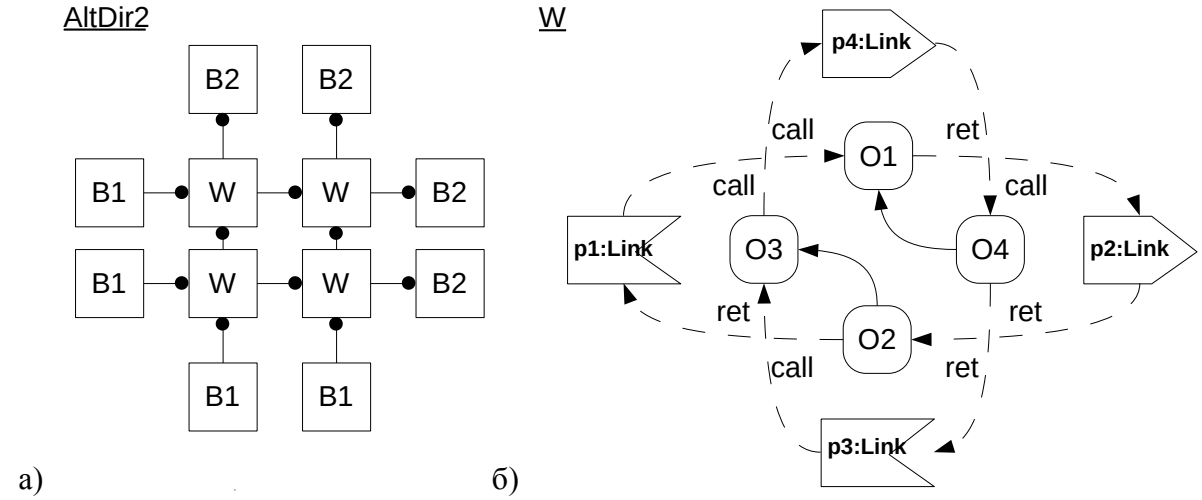

Рис. 7. Композиция процессов (а) и рабочий процесс (б) в методе переменных направлений с декомпозицией по данным.

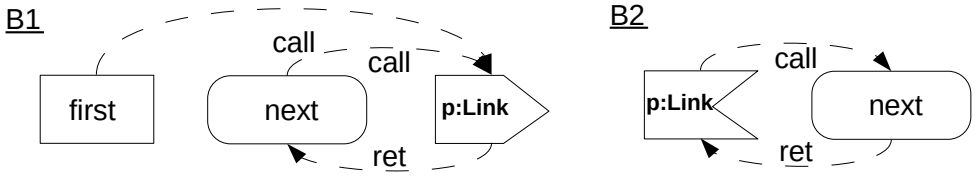

Рис. 8. Граничные процессы в методе переменных направлений с декомпозицией по данным.

Протокол канала для данного вида декомпозиции аналогичен рис. 6 (а). Граничные процессы на сетке рис. 8 используются для запуска прямых и обратных ходов прогонок соответственно. Основу алгоритма составляют рабочие процессы сетки рис. 7 (б). Метод O3 отвечает за выполнение прямого хода вертикальной прогонки, а O1 – прямого хода горизонтальной прогонки. Метод O4 отвечает за выполнение обратного хода вертикальной прогонки, а O2 – обратного хода горизонтальной прогонки.

Правильная очередность запусков методов в процессе рис. 7 (б) организуется следующим образом. В каждом процессе хранится номер текущего направления прогонки. У каждого метода имеется предусловие, проверяющее можно ли начать обработку. Если обработку можно начать при запуске метода, то изменяется номер текущего направления на следующее, выполняется прогонка по текущему направлению и возвращается значение «истина». Если нельзя начать обработку, метод возвращает значение «ложь». Кроме этого в начале каждого цикла прогонок проверяется условие остановки. Оно может быть реализовано как предусловие метода next рис. 8.

3.3. Вычисление с портфелем задач

Типовыми схемами решения многих переборных и оптимизационных задач являются схемы «управляющий - рабочие». Рассматривается модель одной из общих схем управления вычислениями из данного класса: «вычисление с портфелем задач». Это схема с активными рабочими, реализующая управление pull-типа. Здесь иллюстрируется описание более сложного протокола взаимодействия процессов (по сравнению с протоколами «запрос-ответ») с несколькими типами сообщений.

На рис. 9 показана композиция процессов схемы «портфель задач». Обработка заданий выполняется по инициативе рабочих процессов w1,…, wN, что видно из направления каналов p->p1:Link,…, p->pN:Link от рабочих к управляющему процессу.

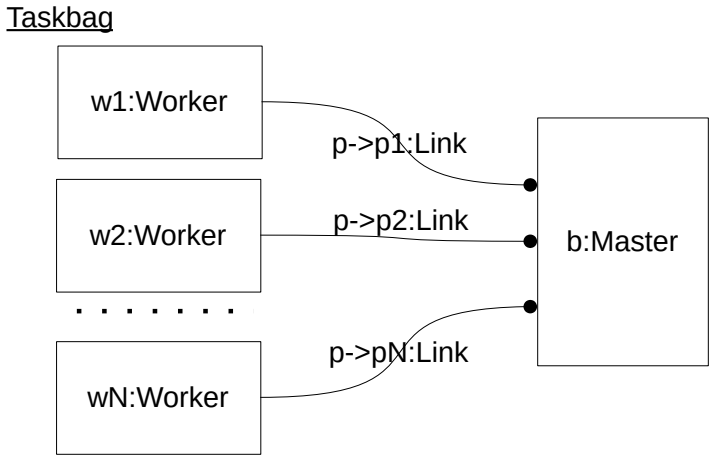

Рис. 9. Композиция процессов при вычислении с портфелем задач.

Роль рабочих процессов рис. 10 (б) заключается в периодическом получении задания от управляющего процесса в сообщении task и отправки результата его обработки result. Обработка выполняется в методе do. Цикл обработки начинается по инициативе рабочего с отправки сообщения start и заканчивается при получении от управляющего процесса сообщения stop. Если потребуется, в методе stop можно выполнить операцию очистки.

На рис. 11 показан фрагмент управляющего алгоритма, реализованного в процессе Master и относящийся к порту p_N. Методы процесса get, put, if_task и start реализуют специфические для решаемой задачи операции: get – извлечение очередной задачи; put – сохранение результата; if_task – проверку наличия задачи для обработки. Метод start может использоваться для инициализации.

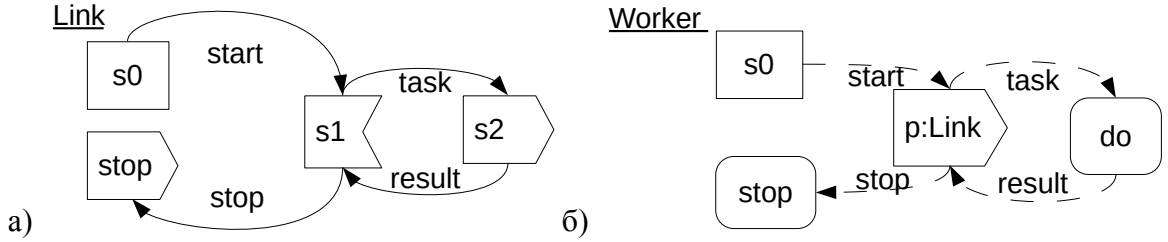

Рис. 10. Протокол канала (а) и рабочий процесс (б) алгоритма вычисления с портфелем задач.

Методы if_active, $wait_N$, $stop_N$, $waiting_N$ являются универсальными для произвольного численного метода. Метод if_active возвращает истинное значение, если имеются активные рабочие процессы. То есть, имеются каналы, не находящиеся в состоянии «доставлено сообщение start или result», что в программной реализации определяется простым подсчетом при поступлении и отправке сообщений. Метод $wait_N$ устанавливает признак, что канал N находится в состоянии ожидания заданий, метод $waiting_N$ возвращает значение данного признака. Метод $stop_N$ используется при отправке сообщения stop рабочему для начала очистки.

Остановка вычислений и очистка начинаются, если нет задач и нет активных рабочих процессов. Когда сформирована и отправлена очередная задача, происходит поиск ожидающего рабочего процесса. Если такой процесс найдется и есть еще одна задача для обработки, то выполняется ее формирование и отправка рабочему. В противном случае управляющий процесс либо заканчивает обработку сообщения и ждет ответа от рабочих, либо начинает остановку вычислений и очистку.

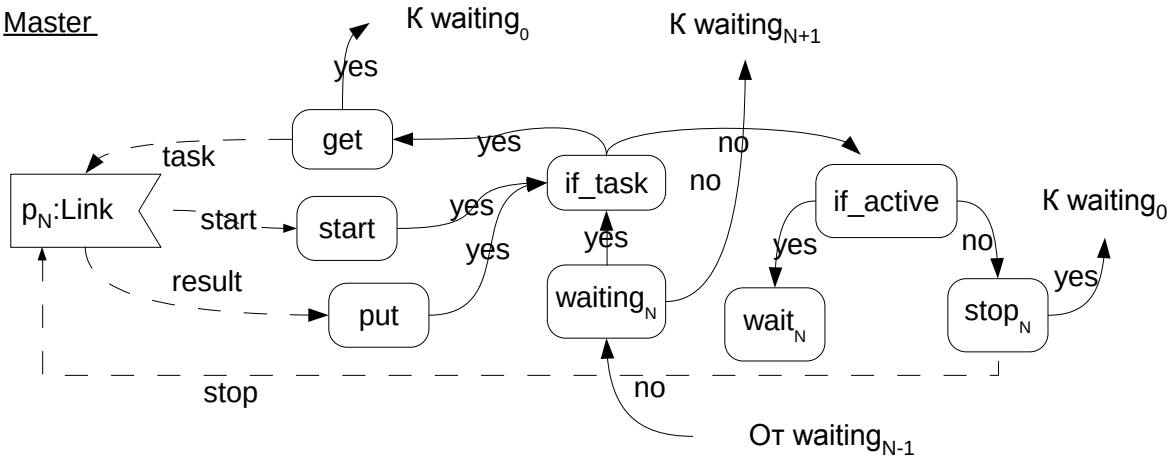

Рис. 11. Фрагмент диаграммы управляющего процесса для алгоритма вычисления с портфелем задач.

3.4. Вычислительный конвейер

Пример вычислительного «конвейера» иллюстрирует использование линейной регулярной структуры процессов. Он может применяться как в алгоритмах с функциональной декомпозицией, так и в алгоритмах с декомпозицией по данным. Показан способ инициализации и остановки вычислений в предлагаемой нотации.

Композиция процессов конвейера рис. 12 – это цепочка из процессов типа Center, замкнутая с концов процессом типа Left и процессом типа Right. Особенностью инициализации вычислений в конвейере является то, что все его ступени, кроме r:Right, должны получить сообщение ret. Это достигается посылкой сообщения ret самому себе из методов first, что видно на рис. 13 (в), рис 13 (а), рис. 14. В методах first также может выполняться инициализация.

Остановкой вычислений управляют методы next процессов. Для остановки требуется, чтобы метод вернул значение «ложь». Это приводит к тому, что не отправляются исходящие сообщения: ret – процессу слева и call – процессу справа. При остановке процессы выходят из вычислений от крайнего левого к крайнему правому.

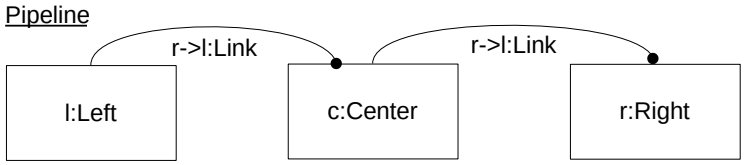

Рис. 12. Композиция процессов конвейера.

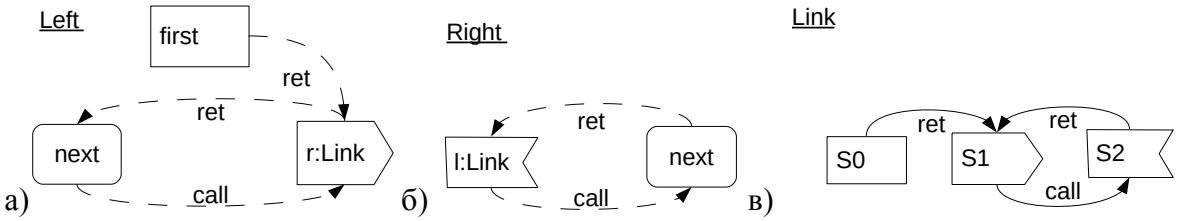

Рис. 13. Левый (а) и правый процессы (б) и протокол канала (в) конвейера.

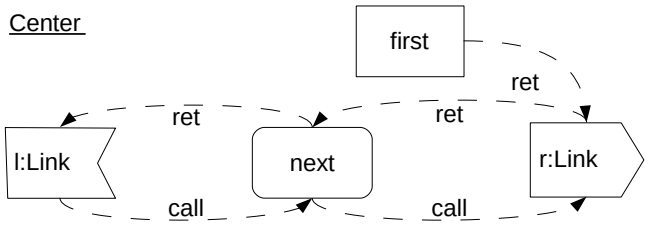

Рис. 14. Центральный процесс конвейера.

4. Результаты

В новой версии системы Templet, представленной в работе, нам удалось упростить графическую нотацию [6] за счет сокращения числа графических примитивов до 3х типов связей и 4х типов вершин в аннотированных орграфах моделей. Было упрощено и структурировано описание семантики исполнения путем выделения групп правил для запуска метода, отправки исходящих сообщений и переходов управления. Построена

библиотека примеров моделей параллельных алгоритмов, демонстрирующих практическое применение нотации Templet.

Для автоматизации построения моделей в нотации Templet нами разработаны расширения для редакторов диаграмм Draw из пакета Open Office и Visio из пакета Microsoft Office. Новый формат диаграмм поддерживается в созданном ранее трансляторе Graphplus templet (доступен для загрузки на сайте graphplus.ssau.ru), что позволяет получить код параллельной программы непосредственно по ее модели.

Совместно со специалистами прикладных областей, используя нотацию Templet, выполнена разработка параллельных алгоритмов решения задач из области нелинейной динамики (параллельный алгоритм метода переменных направлений с функциональной декомпозицией) [7] и компьютерной оптики (параллельный алгоритм решения уравнения Максвелла для плоской волны по схеме конвейера) [8]. Эффективность алгоритмов проверена на персональных многоядерных системах и суперкомпьютере «Сергей Королев» Самарского государственного аэрокосмического университета [9].

СПИСОК ЛИТЕРАТУРЫ

1. Process Description Capture Method (доступ http://www.idef.com/IDEF3.htm, дата обращения 30.05.2012).
2. Unified Modeling Language (доступ http://www.omg.org/spec/UML/, дата обращения 30.05.2012).
3. The Ptolemy Project (доступ http://ptolemy.eecs.berkeley.edu/, дата обращения 30.05.2012)
4. *Шеер А.-В.* ARIS - моделирование бизнес-процессов. Вильямс, 2000. 175 с.
5. *Востокин С.В.* Визуальное моделирование в разработке параллельных алгоритмов. Метод и программные средства. LAMBERT Academic Publishing, 2011. 304 с.
6. *Востокин С.В.* Система автоматизации параллельного программирования Graphplus templet / В сб.: Параллельные вычисления и задачи управления: Тр. Пятой Международн. конф-ции. М.: ИПУ РАН, 2010. С. 1143-1156.
7. *Курушина С.Е.* Аналитическое исследование и численное моделирование контрастных диссипативных структур в поле флуктуаций динамических переменных // Известия ВУЗов. Прикладная нелинейная динамика. 2009. №6. С. 125-138.
8. *Головашкин Д.Л, Сойфер В.А.* Анализ прохождения электромагнитного излучения через дифракционную линзу // Автометрия. 1999. Вып.6. С.119-121.
9. *Востокин С.В., Хайрутдинов А.Р., Литвинов В.Г.* Программный комплекс параллельного программирования Graphplus templet // Вестн. Сам. гос. техн. ун-та. Сер. Физ.-мат. науки. 2011. №4(25). С. 146–153.