# ИСПОЛЬЗОВАНИЕ РАСШИРЯЕМЫХ ЯЗЫКОВ ДЛЯ ПРОГРАММИРОВАНИЯ МНОГОПОТОЧНЫХ ПРИЛОЖЕНИЙ

***А.П. Свиридов***

*Южно-Уральский госуниверситет, Челябинск*

**Введение**

Одним из наиболее распространенных на сегодняшний день средств создания параллельных приложений является технология OpenMP [2, 6]. Эта технология предоставляет пользователю набор простых, интуитивно понятных директив для «разметки» кода на параллельные и последовательные части. По сути, директивы предоставляют пользователю удобный синтаксис, позволяющий программисту писать более декларативный код. Этот подход к программированию параллельных приложений оказался настолько удобен, что сравнительно недавно анонсированная технология OpenACC полностью его копирует.

Однако в этом подходе присутствует один существенный недостаток. Дело в том, что технология подобная OpenMP не может быть реализована посредством обычной библиотеки. Поддержка данной технологии конкретным компилятором полностью ложится на плечи разработчиков этого компилятора. Это существенно затрудняет процесс ее внедрения и изменения. Так, например, технология OpenACC сейчас поддерживается исключительно тремя коммерческими компиляторами, что по понятным причинам значительно ограничивает круг людей, которые могут ее применять и изучать.

Однако существуют языки программирования, в которых программист может добавлять новые специальные конструкции без изменения компилятора – расширяемые языки.

*Расширяемый язык* – это такой язык программирования, синтаксис и семантика которого могут быть изменены и расширены для облегчения написания программы, но при этом расширение должно выполняться без изменения компилятора языка. В настоящее время существует небольшое количество расширяемых языков программирования, самыми популярными из которых являются диалекты языка LISP.

По сравнению с традиционными языками программирования, расширяемые языки обладают рядом преимуществ. Строить системы программирования на основе расширяемых языков проще, и они по определению являются открытыми. В рамках расширяемого языка можно строить многоуровневые модели программирования. Программист получает, с одной стороны, больший контроль над используемыми преобразованиями программ, а с другой – возможность программировать на низком уровне в случае необходимости. Вследствие открытости системы программист получает возможность использовать преобразования, полезные только для специфических задач, и которые не имеет смысл реализовывать в самом компиляторе [5].

**2. Библиотека CLiMP**

В рамках данной работы ведется разработка библиотеки параллельного программирования CLiMP для языка программирования Common Lisp [3, 4], предоставляющей программисту инструменты, аналогичные стандарту OpenMP, но использующие механизм расширений для предоставления нового синтаксиса.

В отличие от технологии OpenMP, в библиотеке CLiMP большое внимание уделяется возможности расширения пользователем существующего функционала. Более того, при разработке библиотеки изначально были созданы инструменты, которые позволяют вносить добавления, а уже с их помощью была реализована большая часть библиотеки.

Так, например, в библиотеке CLiMP предусмотрена команда `define-parallel-command`, предоставляющая пользователю возможность с помощью специальных правил объявлять новые команды параллельного региона. С помощью `define-parallel-command` были реализованы параллельные аналоги циклов, средства синхронизации потоков (критические секции, барьерная синхронизация), средства контроля над потоком управления и др.

Еще одним примером инструмента расширения является команда `define-reduction`. С помощью данной команды пользователь может определить правила распределения некоторой структуры данных между потоками и правила сбора локальных данных обратно в глобальную по отношению к потокам переменную. С помощью данной команды были определены все варианты `reduction`, предоставляемые стандартом OpenMP.

На рис. 1 приведен пример «Hello world» программы с использованием библиотеки CLiMP. Конструкция `parallel` является аналогом директивы `#pragma omp parallel`. С ее помощью пользователь может создать параллельный регион и сконфигурировать его с помощью различных параметров (`number-of-threads`, `reduction`, `private` и др.) и внутренних команд (`single`, `barrier`, `master` и др.).

```
(parallel (:number-of-threads N)
 (format t "~&Hello from: ~a" thread-number)
 (single
  (format t "~&Number-of-threads: ~a" number-of-threads)))
```

Рис. 1. «Hello world» программа с использованием библиотеки CLiMP

Библиотека CLiMP также способствует пошаговому процессу создания параллельной программы. На рис. 2 приведен пример использования стандартного цикла dotimes и его параллельного аналога. В параллельных аналогах циклов доступны все параметры и внутренние команды конструкции parallel.

```
(dotimes (var niterations)    => (parallel-dotimes (var
niterations)
  ...)                        =>   ...)
```

Рис. 2. Перевод однопоточной версии цикла к многопоточной

### 2.1. Архитектура библиотеки

Основным механизмом создания расширений является макрос [1]. *Макрос* – это аналог функции, которая исполняется на этапе компиляции (либо до него), принимает на вход фрагменты кода и возвращает новый фрагмент кода. После исполнения макроса возвращаемый фрагмент кода подставляется на место вызова макроса. При этом макросы в языке Common Lisp могут использовать весь функционал, уже реализованный в системе. Они могут использовать как пользовательские, так и стандартные функции, макросы, переменные и константы.

Примером эффективного использования макросов является CLOS (Common Lisp Object System). CLOS – впервые была реализована именно как сторонняя библиотека, однако она предоставляла пользователю полноценный синтаксис для создания объектно-ориентированных приложений.

На рис. 3 представлена общая архитектура библиотеки. Прямоугольниками на схеме изображен функционал, который реализован с помощью функций, а шестигранниками с помощью макросов.


Рис. 3. Архитектура библиотеки CLiMP

Как мы видим, с помощью функции реализована лишь низкоуровневая работа по пересылке данных в пул потоков. Важно, что с помощью подобной функции и набора средств синхронизации (мьютексов, переменных состояния) можно «эмулировать» функционал предоставляемый, например, стандартом OpenMP: используя мьютекс, переменную состояния и счетчик можно реализовать барьерную синхронизацию, зная номер потока, можно организовать цикл со статическим распределением итераций и т. д.

Другими словами, можно вывести набор правил, с помощью которых может быть осуществлен перевод декларативной конструкции (например, директивы OpenMP) в код, реализующий ее функционал.

В библиотеке CLiMP подобный перевод осуществляется в рамках конструкции `parallel`. Важно, что набор таких правил не фиксирован и пользователь может вводить в библиотеку новые высокоуровневые конструкции параллельного программирования.

### 2.2. Результаты проведения экспериментов

В ходе выполнения работы проведено тестирование производительности компонентов разрабатываемой библиотеки. В качестве одного из экспериментов выбрана задача построения графического представления множества Мандельброта.

Решение данной задачи выполнено в двух вариантах: с использованием языка C (Intel C Compiler 12.1.0) и технологии OpenMP и с использованием языка Common Lisp (Steel Bank Common Lisp 1.0.56) и разработанной библиотеки CLiMP.

Тестирование проводилось на узле суперкомпьютера «СКИФ-Аврора ЮУрГУ». Результаты тестирования приведены в табл. 1 и 2 и на рис. 4 и 5.

Таблица 1. Время выполнения программы в секундах в зависимости от количества потоков

| | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **ICC + OpenMP** | 13,85 | 6,98 | 4,72 | 3,55 | 2,97 | 2,49 | 2,16 | 1,91 | 1,71 | 1,57 | 1,42 | 1,31 |
| **SBCL + CLiMP** | 18,38 | 9,28 | 6,25 | 4,73 | 3,94 | 3,32 | 2,89 | 2,54 | 2,28 | 2,07 | 1,89 | 1,75 |

Таблица 2. Ускорение работы программы в зависимости от количества потоков

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **ICC + OpenMP** | 1,00 | 1,98 | 2,93 | 3,90 | 4,66 | 5,56 | 6,41 | 7,25 | 8,10 | 8,82 | 9,79 | 10,61 |
| **SBCL + CLiMP** | 1,00 | 1,98 | 2,94 | 3,89 | 4,67 | 5,55 | 6,37 | 7,25 | 8,08 | 8,88 | 9,72 | 10,53 |

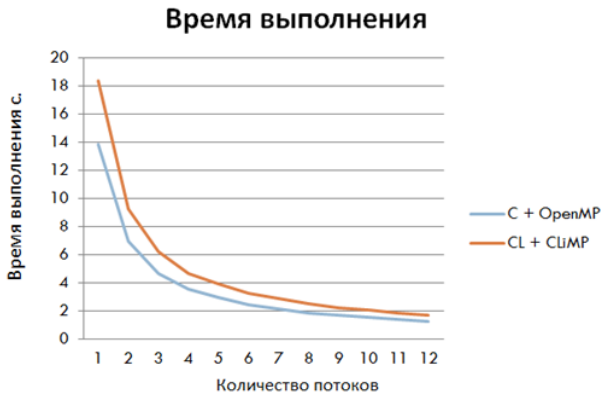


Рис. 4. Время выполнения программы

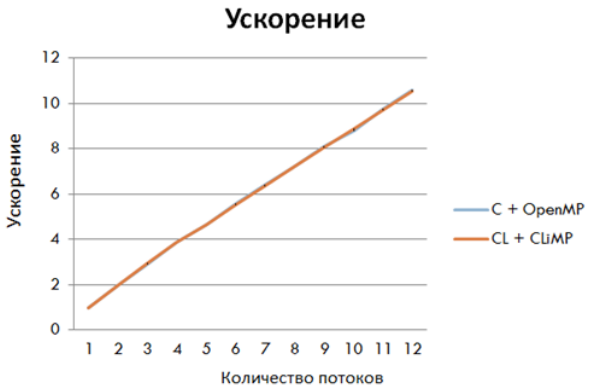


Рис. 5. Ускорение работы программы

Как видно из полученных результатов, несмотря на то, что CLiMP – это просто библиотека, реализованная стандартными средствами языка Common Lisp, она позволяет получать ускорение приблизительно равное ускорению программы, использующей технологию OpenMP.

### 3. Заключение

В статье рассказано о парадигме расширяемого программирования, ее преимуществах и возможностях использования для программирования параллельных вычислений, описана архитектура библиотеки CLiMP, расширяющей язык Common Lisp для программирования многопоточных вычислений. Разработанная библиотека протестирована на ряде задач, для которых показала увеличение производительности в несколько раз без существенного увеличения объема исходного кода. При этом если бы соответствующий код писался вручную, его объем вырос бы в несколько раз, а читабельность программы значительно снизилась.

### Литература


1. Graham P. On Lisp: Advanced Techniques for Common Lisp. Prentice Hall, 1993. 413 p.
2. OpenMP Version 3.1 Complete specification. – [http://www.openmp.org/mp-documents/OpenMP3.1.pdf].
3. Seibel P. Practical Common Lisp. Apress, 2005. 500 p.
4. Steele G. Common LISP. The Language. Second Edition. Digital Press, 1990. 1029 p.
5. Адинец А.В.. Использование расширяемых языков для программирования графических процессоров // Параллельные вычислительные технологии (ПаВТ’2011): Труды Международной научной конференции (Москва, 28 марта–1 апреля 2011 г.). Челябинск: Издательский центр ЮУрГУ, 2011. С. 15–26.
6. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. Спб.: БХВ-Петербург, 2002. 606 с.