Использование шаблонного метапрограммирования при реализации параллельных эвристических алгоритмов оптимизации

О.И. Булычов

В статье рассматривается подход к разработке параллельных эвристических алгоритмов оптимизации, реализованный в библиотеке параллельных метаэвристик HeO. Подход основан на использовании шаблонов проектирования, что позволяет получать универсальные и гибкие алгоритмы с возможностью гибридизации. Приводятся примеры реализации параллельного генетического алгоритма и алгоритма имитационной нормализации, и их гибридов.

1. Введение

Оптимизационные задачи возникают в различных областях науки и техники, и для их решения разработан богатый арсенал методов. Однако нахождение точного решения многих задач на практике оказывается невозможным. Это может быть вызвано разными причинами: сложностью решаемой задачи, погрешностями входных данных, временными и аппаратными ограничениями и др. В связи с этим актуальным остается вопрос разработки эффективных приближенных методов решения таких задач.

Благодаря интенсивному развитию компьютерной техники вот уже несколько десятилетий для нахождения приближенного решения многих трудных оптимизационных задач (прежде всего задач дискретной оптимизации) с успехом применяются метаэвристические алгоритмы (см., например, [1]). Популярность данных алгоритмов обусловлена целым рядом факторов, из которых можно отметить следующие два:

- простота и ясность концепций, лежащих в их основе;
- высокий уровень абстракции.

Первый фактор обуславливает относительную простоту реализации метаэвристических алгоритмов на различных программно-аппаратных платформах, а второй позволяет применять данные алгоритмы к широкому кругу оптимизационных задач (как дискретных, так и непрерывных).

Бурное развитие параллельных вычислительных технологий в последние годы способствовало возрождению интереса к метаэвристическим алгоритмам, что обусловлено возможностью их эффективного распараллеливания. Используя различные модели параллельных вычислений в метаэвристических алгоритмах, можно либо улучшать качество получаемых решений, либо ускорять процесс их поиска. На сегодняшний день разработано большое количество программных средств, в которых реализованы те или иные параллельные метаэвристические алгоритмы для решения конкретных задач. Одной из тенденций последнего десятилетия является создание универсальных библиотек[1] параллельных метаэвристических алгоритмов, которые можно применять для решения широкого круга оптимизационных задач.

Кроме того, в последние годы большое внимание уделяется гибридизации метаэвристик, что позволяет на основе базовых алгоритмов получать новые, более эффективные методы оптимизации. Вопросы гибридизации метаэвристических алгоритмов в разных программных продуктах решаются по-разному. Учитывая большое разнообразие метаэвристических алгоритмов, универсального метода их гибридизации просто не существует, однако можно выделить некоторые общие подходы к гибридизации, применяемые на практике. Хороший обзор методов гибридизации метаэвристик приведен в [2].

В настоящей статье рассматриваются практические вопросы программной реализации и гибридизации параллельных метаэвристических алгоритмов в библиотеке HeO, разработанной с использованием современных технологий программирования, метапрограммирования и па-

[1] В англоязычной литературе часто используется термин *frameworks*.

раллельных вычислений на языке C++. Подробное описание библиотеки можно найти в [3]. В следующем разделе будут приведены наиболее важные сведения об этой библиотеке.

2. Библиотека HeO

Библиотека HeO (**He**uristic **O**ptimization) является проектом с открытым исходным кодом и распространяется на основе лицензии MIT. Цель проекта — обеспечить исследователей современными и простыми в использовании средствами для решения широкого круга оптимизационных задач.

Ключевыми особенностями библиотеки являются:

- кроссплатформенность, поддержка архитектур x86 и x64;
- реализация методов оптимизации в виде проблемно-независимых алгоритмических каркасов;
- реализация каждого метода оптимизации с использованием двух технологий параллельного программирования: MPI и OpenMP;
- использование оригинальной обертки (wrapper) для функций MPI;
- наличие мастеров (wizards), облегчающих создание пользовательских проектов;
- прозрачность параллелизма для пользователей библиотеки.

Официальная страница проекта располагается по адресу: http://www.code.google.com/p/heo.[1]

Библиотека представляет собой набор классов, которые можно разделить на две большие группы:

- основные классы;
- вспомогательные классы.

Основные классы участвуют в непосредственной реализации того или иного метода оптимизации. Ядро основных классов библиотеки составляют так называемые *решатели* (solvers). Каждый решатель реализует определенный метод оптимизации в соответствии с определенной технологией параллельного программирования (MPI или OpenMP).

Решатели выполнены в виде шаблонов языка C++ в соответствии с идеологией алгоритмических каркасов. Это означает, что в них:

- реализована только проблемно-независимая часть алгоритма;
- скрыты все аспекты параллельной реализации.

Эти особенности позволяют применять решатели к широкому кругу задач и делают параллелизм прозрачным для пользователя библиотеки. Для использования решателей пользователем должны быть реализованы проблемно-зависимые классы, описывающие конкретную задачу оптимизации и ее решения. Кроме того, в проблемно-зависимых классах обязательно должны быть реализованы методы, специфические для того или иного алгоритма оптимизации.

Вспомогательные классы библиотеки выполняют всевозможные служебные функции (работа с конфигурационными файлами, запуск решателей, передача данных по сети, генерация псевдослучайных чисел и т.п.).

Для демонстрации возможностей библиотеки разработаны несколько проектов, в которых реализованы проблемно-зависимые классы для нескольких задач дискретной и непрерывной оптимизации:

- ONE-MAX — максимизация числа единиц в двоичной строке (тестовая задача для метаэвристических алгоритмов);
- MAX-SAT — задача максимальной выполнимости;
- GridMin — задача минимизации числа покрывающих блоков в #-отношениях недетерминированных конечных автоматов;
- Rastrigin — минимизация функции Растригина;
- Rosenbrock — минимизация функции Розенброка;
- ODE1IVP — численное решение задачи Коши для ОДУ первого порядка.

[1] В настоящий момент для скачивания (в разделе Downloads или через SVN-клиент) доступна версия 1.1 библиотеки.

3. Реализация параллельных метаэвристических алгоритмов и их гибридизация с использованием технологии шаблонного проектирования

В настоящий момент в библиотеке HeO реализованы два метаэвристических метода оптимизации: генетический алгоритм (GA — Genetic Algorithm) и метод имитационной нормализации или имитации отжига (SA — Simulated Annealing), а также их гибрид (GA+SA). Описания последовательных версий методов GA и SA можно найти, например, в [4]. Ниже будет рассмотрен использованный в библиотеке подход к параллельной реализации этих методов, основанный на технологии шаблонного проектирования. Подробное изложение методов шаблонного проектирования приведено в [5].

3.1 Реализация параллельных алгоритмов оптимизации

Как уже было отмечено выше, каждый из методов оптимизации реализован в библиотеке с помощью двух технологий параллельного программирования (MPI и OpenMP). В текущей версии библиотеки основной моделью параллельных вычислений является *мультистартовая* модель[1]. В этой модели алгоритм работает в несколько потоков, независимо решающих задачу с периодической кооперацией.

На рис. 1. представлен псевдокод потоков параллельного генетического алгоритма.

```
сгенерировать начальную популяцию хромосом
вычислить функцию стоимости для каждой хромосомы
while not(выполнено условие остановки) do
  отобрать родительские хромосомы
  выполнить кроссовер (получить потомков)
  применить мутацию к потомкам
  вычислить функцию стоимости для потомков
  сформировать новую популяцию
  if (условие кооперации) then выполнить кооперацию
end do
Выход: лучшие хромосомы (решение задачи)
```

Рис. 1. Псевдокод потоков параллельного генетического алгоритма

Кооперация в параллельном GA заключается в периодическом обмене выбранными с помощью некоторой процедуры решениями и фактически является аналогом миграции в островной модели GA.

Псевдокод потоков параллельного метода имитации отжига приведен на рис. 2.

```
сгенерировать начальное решение
вычислить функцию стоимости для начального решения
выбрать начальное значение температуры T
установить счетчик начала итераций k равным 0
while not(выполнено условие остановки) do
  if (условие кооперации) then выполнить кооперацию
  выбрать случайное решение из окрестности текущего
  вычислить его стоимость
  сделать новое решение текущим с вероятностью P(T)
  увеличить счетчик итераций k
  изменить температуру T
end do
Выход: текущее решение
```

Рис. 2. Псевдокод потоков параллельного алгоритма имитации отжига

Кооперация в параллельном SA представляет собой периодический обмен генерируемыми случайными решениями (ходами).

[1] Для генетического алгоритма с использованием технологии OpenMP реализована также модель глобальной популяции.

В библиотеке реализованы два режима кооперации: асинхронный и синхронный, схемы которых приведены на рис. 3.

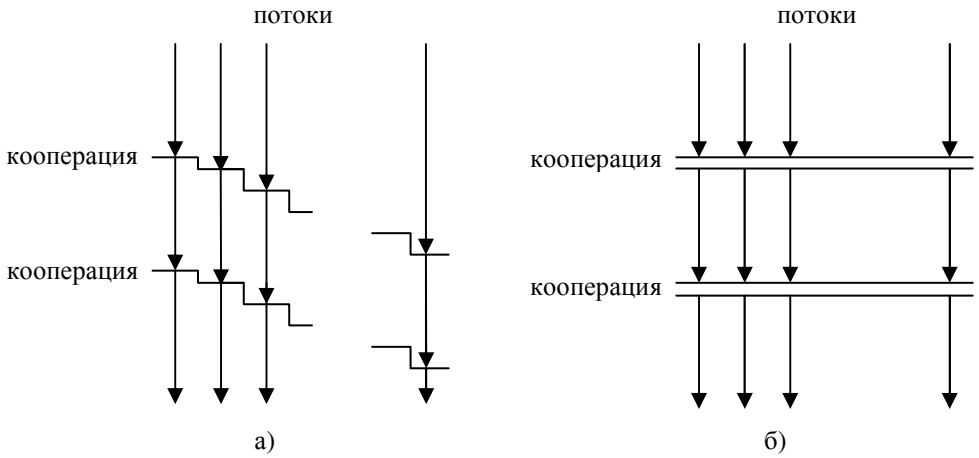

Рис. 3. Схема асинхронной (а) и синхронной (б) кооперации потоков в мультистартовой модели

При синхронной кооперации осуществляется блокирование выполнения потоков и обмен данными между ними через фиксированное число итераций. Асинхронная кооперация инициируется первым потоком через фиксированные промежутки времени и выполняется без блокировки потоков по мере их готовности к обмену данными. Использование синхронной кооперации может приводить к большой нагрузке на сеть в моменты кооперации. Асинхронный режим позволяет эффективно решать задачу на неоднородных кластерах и снизить нагрузку на сеть.

При использовании кооперации возникает проблема *сериализации*, т.е. перевода структур данных программы в последовательность битов и обратно. В большинстве существующих библиотек параллельной оптимизации загрузка, сохранение и передача по сети каждого поля класса производится индивидуально. Это приводит к тому, что при внесении изменений в структуру класса приходится вручную изменять программный код в каждом месте, где требуется сериализация.

С целью упрощения этих операций в библиотеке HeO используются приемы шаблонного метапрограммирования, позволяющие реализовать элементы технологии Reflection (Отражение) для строго типизированного доступа к полям классов. Реализованный алгоритм сериализации базируется на шаблоне проектирования Visitor (Посетитель), позволяющем определять новые операции над классами библиотеки, не изменяя исходный код классов. Для его использования достаточно определить метод `accept()` в требуемых классах, и перечислить в нем необходимые поля класса (см. рис. 4).

```
int                         member1_;
std::vector<float>          member2_;
ref_ptr<some_interface> member3_;
...
template<class V> void accept(V& v)
{
    v(member1_, "member1");
    v(member2_, "member2");
    v(member3_, "member3");
    ...
};
```

Рис. 4. Пример определения метода accept()

В случае если поле само является структурой или классом, сериализация осуществляется рекурсивно. Для некоторых классов, в частности для `std::string`, `std::vector` и классов "умных" указателей, определены особые служебные классы, осуществляющие их сериализацию.

К сожалению, в языке C++ нет встроенного автоматического механизма Reflection. При изменении структуры класса, придется вносить изменение и в функцию `accept()`.

Кооперация потоков в алгоритмах GA и SA реализована с использованием шаблона Strategy (Стратегия). Данный шаблон обеспечивает различные варианты кооперации без изменения исходного кода алгоритмов-клиентов. В настоящий момент реализована стратегия кооперации с обменом данными между потоками по кольцу, однако она может быть заменена на любую другую.

3.2 Гибридизация параллельных алгоритмов

Для разных типов метаэвристических алгоритмов могут быть предложены разные способы гибридизации. В частности, для алгоритмов GA и SA, наиболее популярными способами являются интеграционная (integrative) и кооперационная (cooperative) гибридизация. При интеграционной гибридизации один из алгоритмов является частью другого, например, SA может рассматриваться как один из генетических операторов, выполняемых в основном цикле GA. При кооперационной гибридизации алгоритмы обмениваются друг с другом решениями.

Наибольшие трудности возникают при практической реализации первой стратегии. Как правило, в существующих реализациях гибридного метода GA+SA жестко фиксируется структура алгоритма, а для ее модификации приходится вносить изменения в исходный код программы. Для решения этой проблемы в библиотеке HeO используется шаблон проектирования Factory (Фабрика), позволяющий сделать код создания объектов более универсальным, не привязываясь к конкретным классам, а оперируя лишь общим интерфейсом.

Все операторы гибридного алгоритма наследуются от абстрактного интерфейса `abstract_operator`. В исходном коде гибрида и фабрики имена классов-операторов нигде не упоминаются, что упрощает добавление новых операторов. Каждый оператор регистрируется на фабрике с помощью вспомогательной функции. На рис. 5 показаны отношения между гибридом, фабрикой и операторами.

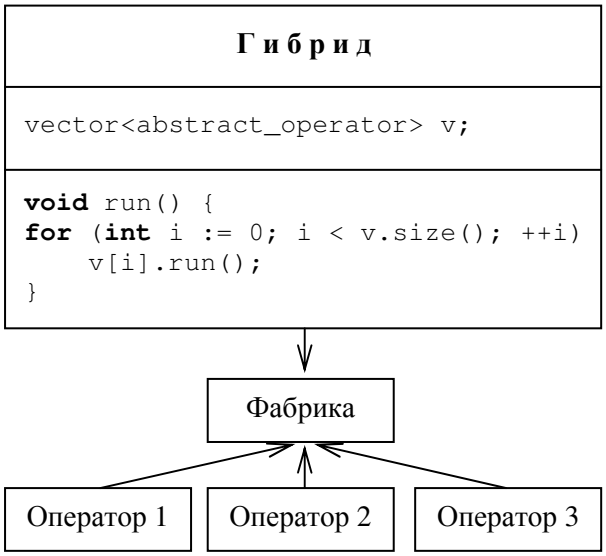

Рис. 5. Диаграмма отношений между гибридом, фабрикой и операторами

Одним из недостатков фабрики считается трудность восстановления *класса* создаваемого ею продукта по его абстрактному интерфейсу, с целью выполнения операции, зависящей от класса (к примеру, сериализации). Реализованный механизм Reflection позволяет эффективно обойти этот недостаток.

Параметры решателей в библиотеке HeO хранятся в конфигурационных файлах в формате xml. При выполнении кода, изображенного на рис. 6, в решателе `solver` будет создан вектор операторов, заданных в xml-файле конфигурации.

```
xml::in in(env.property_tree_);
in >> solver;
```

Рис. 6. Пример инициализации класса

На рис. 7 и 8 приведены фрагменты xml-файлов, задающих различные векторы операторов для генетического алгоритма.

```
<operator>
  <crossover .../>
  <sa .../>
</operator>
```

Рис. 7. Фрагмент конфигурационного файла

В первом случае будет создан вектор операторов [`crossover`, `sa`], а во втором [`sa`, `crossover`, `mutation`], которые будут последовательно применяться на каждой итерации алгоритма.

```
<operator>
  <sa .../>
  <crossover .../>
  <mutation .../>
</operator>
```

Рис. 8. Фрагмент конфигурационного файла

Аналогично с помощью фабрики осуществляется выбор операторов отбора родительских и дочерних хромосом.

Поскольку для правильной работы программы требуется лишь один экземпляр конкретной фабрики, то для ее конструирования используется шаблонный класс Singleton (Одиночка).

При обмене данными между решателями, реализующими различные алгоритмы, может потребоваться необходимость преобразования проблемно-зависимых классов, поскольку представления данных в разных алгоритмах могут быть различными. Для решения этой проблемы в библиотеке HeO используется шаблон проектирования Adapter (Адаптер), позволяющий не принимать во внимание различия между проблемно-зависимыми классами разных алгоритмов.

Такой подход полностью избавляет пользователя от изменения исходного кода гибридного метода и позволяет получать гибридные алгоритмы, наиболее подходящие для решения конкретной задачи.

4. Примеры

В качестве практических примеров использования параллельных эвристических алгоритмов оптимизации рассмотрим две задачи: задачу максимальной выполнимости и задачу поиска минимума функции Розенброка.

4.1 Задача максимальной выполнимости

Задача максимальной выполнимости (MAX-SAT) может кратко быть сформулирована следующим образом: дана конъюнктивная нормальная форма (КНФ) функции N переменных из M клауз. Требуется найти двоичный набор, при котором в этой функции будет выполнено наибольшее число клауз. Клаузы могут быть как фиксированной, так и переменной длины. Описание различных формулировок задач MAX-SAT можно найти, например, в [4].

В данном разделе будут приведены результаты тестирования для задачи ndhf_xits_22_SAT из набора SAT 2009. Параметры задачи: число переменных — 4692, число клауз — 582514. Данная задача является выполнимой, поэтому максимальное число выполнимых клауз совпадает с общим числом и равно 582514.

Задача решалась на системе с общей памятью следующей конфигурации: процессор — Intel Core 2 Quad Q6600 @ 2.4 ГГц; ОЗУ — 4 Гб; операционная система — MS Windows XP

Professional SP 3. Использовалась OpenMP-версия генетического алгоритма, в которой реализована модель глобальной популяции.

Задача решалась с числом потоков $k = 1, 2, 3, 4$. При каждом k выполнялось 10 запусков программы. Для каждого запуска запоминалось время выполнения и число невыполненных клауз. Затем результаты усреднялись.

Основные настройки алгоритма GA приведены на рис. 9.

```
<ga population_size="200"
    offspring_size="100"
    only_offspring="false"
    migration_size="3">
  <parent_selection>
    <tournament tournament_size="16"/>
  </parent_selection>
  <offspring_selection>
    <tournament tournament_size="16"/>
  </offspring_selection>
  <operator>
      <crossover probability="0.7"/>
      <mutation probability="0.002"
                fadeout="false"/>
  </operator>
  <stop_condition max_step="500"/>
  <coop async_mode="true"
        cooperation_rate="25"/>
</ga>
```

Рис. 9. Настройки GA

Результаты решения задачи приведены в таблице 1. Полученные результаты являются приемлемыми для универсальных алгоритмов.

Таблица 1. Результаты решения. N — число потоков, ε — число невыполненных клауз, Δ — процент невыполненных клауз, t — время решения (сек.)

N	**ε**	**Δ**	***t***
1	1100	0,189	143
2	1094	0,188	81
3	1149	0,197	58
4	1128	0,194	44

График ускорения вычислений приведен на рис. 10.

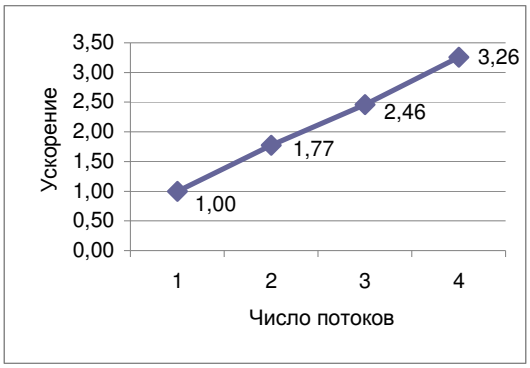

Рис. 10. График ускорения вычислений

4.2 Задача поиска минимума функции Розенброка

Рассмотрим задачу нахождения глобального минимума функции Розенброка n переменных, которая задается следующим образом:

$$f(x)=\sum_{i=1}^{n-1}\left(100\left(x_{i+1}-x_i^2\right)^2+\left(x_i-1\right)^2\right)$$

где $x=(x_1,x_2,...,x_n)$, $x_i\in[-2.048;\,2.048]$. График функции Розенброка для случая двух переменных приведен на рис. 11.

Для этой функции характерно наличие плоского плато, что затрудняет нахождение ее глобального оптимума с помощью генетических алгоритмов (затрудняет сходимость). Точным решением данной задачи является точка $x_0=(1,1,...,1)$, значение функции в которой равно 0.

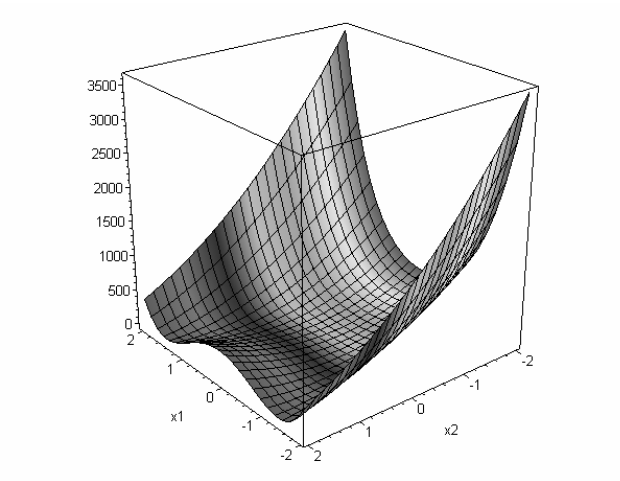

Рис. 11. График функции Розенброка

Данная задача решалась на кластере Томского политехнического университета с помощью MPI-версий обычного генетического алгоритма (GA) и гибрида (GA+SA). В этих версиях реализована островная модель GA. Решение задачи осуществлялось для $n=10$ и $n=50$ с числом потоков 2^k ($k=0,\ 1,\ 2,\ ...,\ 6$). При каждом k выполнялось 10 запусков программы. Фиксировалось время выполнения и стоимость получаемого решения. Для каждого запуска вычислялась невязка найденного решения по формуле $r(x)=\sqrt{\sum_{i=1}^{n}(1-x_i)^2}$. Полученные результаты усреднялись.

Основные настройки алгоритмов GA и GA+SA приведены на рис. 12 и 13.

```
<ga population_size="100"
    offspring_size="200"
    only_offspring="false"
    migration_size="3">
  <parent_selection>
    <roulette/>
  </parent_selection>
  <offspring_selection>
    <tournament tournament_size="5"/>
  </offspring_selection>
  <operator>
      <crossover probability="0.7"/>
      <mutation probability="0.002"
                fadeout="false"/>
  </operator>
  <stop_condition max_step="100000"/>
  <coop async_mode="false"
        cooperation_rate="25"/>
```

```
</ga>
```

Рис. 12. Настройки GA

Как видно из рис. 13, в xml-файле гибридного алгоритма оператор мутации заменен на оператор SA. Операторы `crossover` и `sa` можно было бы переставить местами, и добавить к ним оператор `mutation`, получив при этом новый гибридный алгоритм, не изменяя программного кода.

```
<ga population_size="50"
    offspring_size="100"
    only_offspring="false"
    migration_size="3">
  <parent_selection>
    <roulette/>
  </parent_selection>
  <offspring_selection>
    <tournament tournament_size="3"/>
  </offspring_selection>
  <operator>
    <crossover probability="0.7"/>
    <sa probability="0.03"
        move_probability="0.02"
        initial_temperature="1.5"
        cooling_rate="0.994"
        heating_rate="1.2"
        isotherm_moves="10"
        update_policy="0">
        <stop_condition max_step="100000"/>
    </sa>
  </operator>
  <stop_condition max_step="40"/>
  <coop async_mode="false"
        cooperation_rate="25"/>
</ga>
```

Рис. 13. Настройки GA+SA

В табл. 2 приведено среднее время решения задачи обоими методами в зависимости от числа потоков и n.

Таблица 2. Среднее время решения, сек.

Число потоков	**GA**		**GA+SA**	
	***n* = 10**	***n* = 50**	***n* = 10**	***n* = 50**
1	6,555023	18,92139	2,402968	9,61315
2	6,606100	18,92139	2,577363	10,79885
4	7,029273	18,97877	2,623410	11,54005
8	6,704668	19,63942	2,738245	11,54005
16	7,873530	19,15085	2,890340	11,34775
32	6,843558	20,72200	2,890340	11,66562
64	8,593803	19,35119	3,112738	13,45320

Так как число особей в популяции для каждого потока оставалось неизменным (равным 50), то происходит не ускорение вычислений, а повышение качества получаемого решения. На рис. 14 и 15 приведены графики зависимости невязки решения от числа потоков для каждого метода при $n = 10$ и $n = 50$.

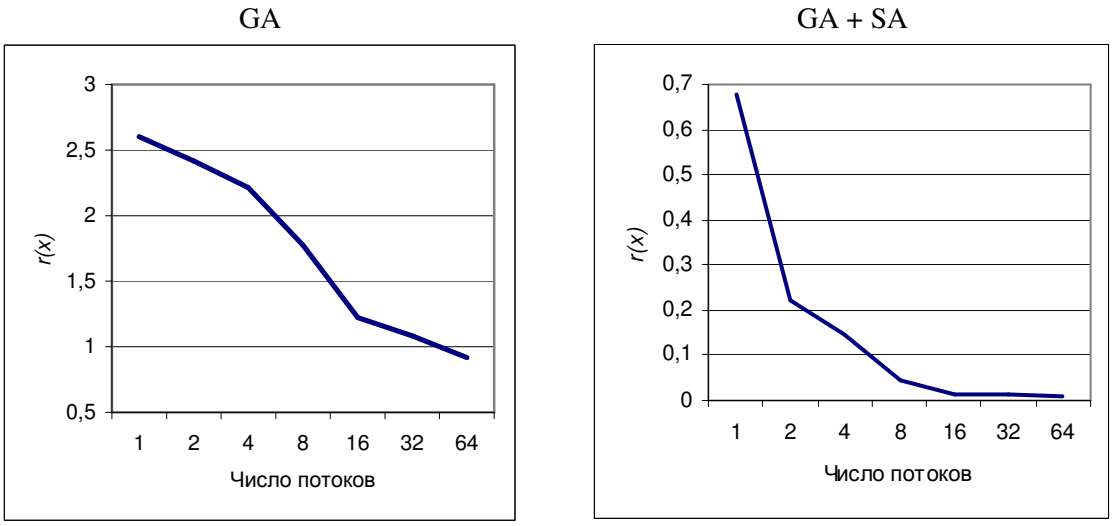

Рис. 14. Графики зависимости невязки решения от числа потоков при $n = 10$

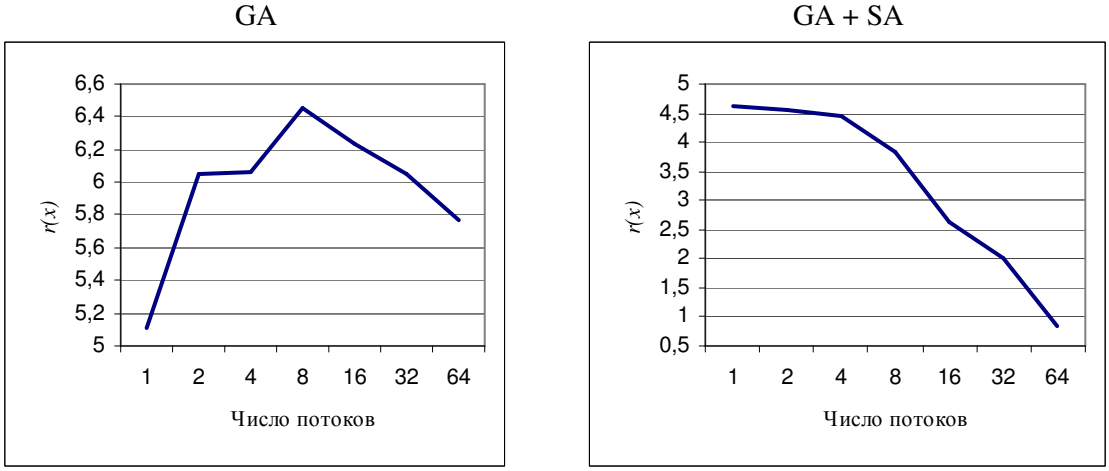

Рис. 15. Графики зависимости невязки решения от числа потоков при $n = 50$

Из приведенных таблицы и графиков видно, что гибридный алгоритм позволяет за меньшее время получить для данной задачи гораздо более точное решение.

5. Заключение

Рассмотренный в работе подход к реализации параллельных эвристических алгоритмов и их гибридов, основанный на использовании технологии шаблонного проектирования, отличается универсальностью и высокой гибкостью и может быть использован при разработке параллельных алгоритмов других типов. Данный подход позволяет:

- повысить возможность повторного использования исходного кода;
- повысить наглядность программы;
- облегчить модификацию программы;
- упростить создание гибридных алгоритмов.

Однако использование такого подхода требует высокой квалификации программиста и знакомство с основами технологии шаблонного проектирования.

В настоящее время в рамках библиотеки HeO планируется реализация уже имеющихся методов с использованием технологии параллельного программирования Intel Threading Building Blocks (TBB).

Автор выражает благодарность руководству и сотрудникам суперкомпьютерного кластера «СКИФ-политех» Томского политехнического университета.

Литература

1. Handbook of metaheuristics / Ed. by F. Glover, G. A. Kohenberger. – Kluwer Academic Publishers, 2003.
2. Raidl G.R., A unified view on hybrid metaheuristics // Proceedings of Hybrid Metaheuristics, Third International Workshop, volume 4030 of LNCS / Ed. by F.Almeida et al. – Springer, 2006. – P. 1-12.
3. Цыганов А.В., Булычов О.И. НеО: библиотека метаэвристик для задач дискретной оптимизации // Программные продукты и системы. – 2009. – № 4. – С. 148-151.
4. Громкович Ю. Теоретическая информатика, 3 изд.: Пер. с англ. – СПб.: БХВ-Петербург, 2009. – 360 с.
5. Влиссидес Дж., Джонсон Р., Гамма Э., Хелм Р. Приемы объектно-ориентированного проектирования. Паттерны проектирования, – СПб.: Питер, 2007 г.