УДК 004.023+ 004.421.2+ 519.178 Дата подачи статьи: 31.08.22, после доработки: 14.09.22
DOI: 10.15827/0236-235X.140.583-597 2022. Т. 35. № 4. С. 583–597

Программное средство GraphHunter поиска отображения параллельной программы на структуру суперкомпьютерной системы

А.В. Баранов [1], *к.т.н., доцент, ведущий научный сотрудник, antbar@mail.ru, abaranov@jscc.ru*
Е.А. Киселев [1], *к.т.н., старший научный сотрудник, e.a.kiselev@yandex.ru, kiselev@jscc.ru*
П.Н. Телегин [1], *к.т.н., ведущий научный сотрудник, pnt@jscc.ru*
А.А. Сорокин [2], *студент, rexantmaster@yandex.ru*

[1] *Межведомственный суперкомпьютерный центр РАН – филиал ФГУ ФНЦ НИИСИ РАН, г. Москва, 119334, Россия*
[2] *МИРЭА – Российский технологический университет, г. Москва, 119454, Россия*

Одной из известных задач в области высокопроизводительных вычислений является поиск оптимального отображения процессов параллельной программы на узлы суперкомпьютерной системы. Решение такой задачи позволяет минимизировать накладные расходы на информационные обмены между выполняющимися процессами и, как следствие, повысить производительность вычислений. При поиске отображения суперкомпьютерная система и параллельная программа представляются в виде графов.

В настоящей статье задача отображения решается применительно к системе коллективного пользования суперкомпьютером, которая ведет очередь параллельных программ. После прохождения очереди параллельной программе выделяется новое, заранее неизвестное подмножество узлов суперкомпьютера. В этих условиях необходимо за приемлемое время построить граф выделенного подмножества узлов и найти подходящее отображение параллельной программы на этот граф. При поиске отображения предлагается задействовать параллельные алгоритмы, для выполнения которых использовать выделенные узлы суперкомпьютера.

Для исследования свойств алгоритмов отображения было разработано программное средство GraphHunter, позволившее провести эксперименты с тремя параллельными алгоритмами: имитации отжига, генетическим алгоритмом и их комбинацией. В настоящей статье рассмотрена структура разработанного программного средства GraphHunter, приведены результаты экспериментов с запусками GraphHunter на суперкомпьютере МВС-10П ОП.

Ключевые слова: *высокопроизводительные вычисления, параллельные алгоритмы отображения, имитация отжига, генетический алгоритм, планирование заданий.*

Современный суперкомпьютер представляет собой параллельную систему из вычислительных узлов, соединенных высокоскоростной сетью. Топология сети может быть разной и включать линии связи разной пропускной способности и латентности. Суперкомпьютерную систему можно представить в виде графа $G_s = (B, E_s)$, где B – множество узлов; E_s – множество дуг, представляющих линии связи между узлами. Производительность линии связи характеризуется весами дуг графа m_{ij}, $(i, j = 1, \ldots, N)$, $(i, j) \in E_s$.

Основное назначение суперкомпьютера – выполнение прикладных параллельных программ. Параллельная программа представляет собой совокупность процессов, обменивающихся между собой информацией с различной интенсивностью. Параллельная программа представляется графом $G_p = (A_p, E_p)$, где A_p – множество вершин, соответствующее процессам; E_p – множество дуг, представляющих информационные связи между процессами. Количество процессов $N = |A_p|$. Дугам графа приписываются веса c_{ij} $(i, j = 1, \ldots, N)$, отражающие интенсивность информационного обмена между i-м и j-м процессами. Данный граф будем называть программным, или информационным графом.

Отображение информационного графа параллельной задачи G_p на структуру суперкомпьютера, заданной графом G_s, обозначается $\varphi : A_p \to B$ и представляется матрицей

$X = \{X_{ij} : i \in A_p, j \in B\}$, где $X_{ij} = 1$, если $\varphi(i) = j$, и $X_{ij} = 0$, если $\varphi(i) \neq j$. Критерием оптимальности отображения (при условии равной производительности всех вычислительных узлов) служит некоторый функционал $F(X)$ [1]:

$$F(X) = \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{p=1}^{N}\sum_{k=1}^{N} m_{ij}c_{kp}X_{ki}X_{pj} \to \min. \qquad (1)$$

Решение задачи требуется обеспечить в системе коллективного пользования. На вход системы поступает поток пользовательских заданий, из которых формируется очередь. Каждое задание включает параллельную программу, для выполнения которой требуется подмножество узлов системы. При запуске очередного задания узлы для него выделяются из числа свободных на этот момент, то есть какие конкретно узлы будут выделены заданию, заранее неизвестно. Для каждого задания требуется найти оптимальное отображение программного графа на заранее неизвестный граф подмножества выделенных узлов в соответствии с (1). Отображение должно строиться за приемлемое время, то есть оно должно быть существенно меньше времени выполнения задания (в среднем несколько часов) и не превышать при этом время системных таймаутов (до 5–15 минут).

В работе [2] для системы управления заданиями суперкомпьютера МВС-1000 [3], установленного в ИПМ им. М.В. Келдыша РАН, предложен двухэтапный метод РГО (Разрезание–Генетика–Отжиг) отображения программного графа на граф вычислительной системы. На первом этапе для очередного задания выбираются узлы суперкомпьютера из числа свободных на момент запуска. Выбор узлов производится с помощью модифицированного алгоритма нахождения минимального разрезания графа [2, 4], который позволяет выделить для задания множество наиболее связанных узлов из числа свободных. На втором этапе осуществляется непосредственное отображение программного графа на граф подсистемы из выбранных узлов. Для нахождения отображения используются выбранные узлы, на которых перед запуском задания работает параллельный алгоритм моделирования отжига с генетическими операциями ПОГ (Параллельный–Отжиг–Генетика). Параллельная схема алгоритма, основанного на сочетании имитационного отжига и генетического алгоритма, позволила получать отображения высокой точности за короткое время. Апробация алгоритма ПОГ осуществлялась на системе МВС-1000, и для ряда приложений удалось получить прирост быстродействия до 45 %. Система МВС-1000 состояла всего из 32 однопроцессорных узлов.

В работе [5] на суперкомпьютере МВС-10П, установленном в Межведомственном суперкомпьютерном центре РАН, удалось решить задачу отображения для теста HPCG. Для отображения был применен параллельный алгоритм имитации отжига PCGA, идея которого заключается в локализации коммуникационного трафика интенсивно обменивающихся ветвей программы по сильносвязанным узлам суперкомпьютера.

В данной статье рассматривается продолжение работ [2, 5] с целью создания программного средства для экспериментального исследования характеристик алгоритмов поиска отображения параллельных программ на структуру суперкомпьютера.

Методы, алгоритмы и средства решения задачи отображения

Для нахождения оптимального отображения применяются различные методы, алгоритмы и программные средства. Классификация известных алгоритмов отображения приведена на рисунке 1.

Точные алгоритмы всегда находят минимальное значение целевой функции (1). К ним относятся полный перебор всех возможных отображений и метод ветвей и границ. Поскольку задача поиска отображения является NP-полной, время работы точных алгоритмов для графов средних и больших размеров может оказаться неприемлемо большим. Сокращение времени до приемлемых величин возможно за счет применения приближенных алгоритмов. Эти алгоритмы в общем случае находят субоптимальное отображение, обеспечивая близкое к минимальному значение целевой функции (1). Среди приближенных алгоритмов выделяют графовые и эвристические алгоритмы. Для ускорения решения задачи часто проводят распараллеливание алгоритмов.

Активная разработка методов, алгоритмов и средств отображения ведется с 80-х годов прошлого века. В настоящее время этой проблеме посвящено множество научных публикаций.

Графовые алгоритмы сравнительно сложно распараллеливаются, часто возникает необходимость улучшения полученных решений с помощью других методов оптимизации.

В работе [6] авторы за счет построения отображения программного графа оптимизи-

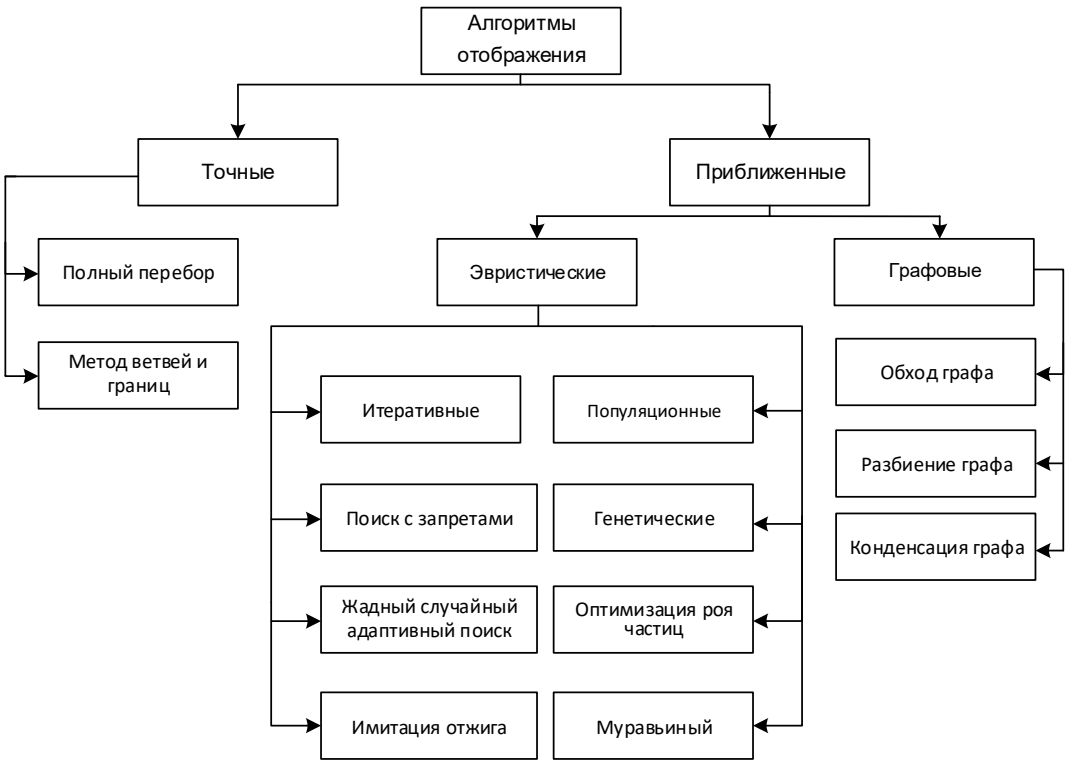

Рис. 1. Классификация алгоритмов отображения

Fig. 1. A classification of mapping algorithms

руют такие параметры, как средняя загруженность канала передачи данных и общее количество переходов, выполняемых сообщениями. Авторы разработали жадный алгоритм поиска отображения, основанный на обходе графа в ширину с последующим улучшением результатов с помощью алгоритма Кернигана–Лина и алгоритма обхода графа в глубину. При решении задачи умножения разряженных матриц на вектор на суперкомпьютере IBM Blue Gene прирост производительности составил 23 %.

Для рекурсивного разбиения графа можно использовать библиотеку Scotch [7], которая активно применяется исследователями. Другая известная библиотека – LibTopoMap [8], в ней реализован ряд алгоритмов отображения, в том числе жадный алгоритм поиска самого загруженного процесса в коммуникационном плане и вычислительного элемента, имеющего наибольшую коммуникационную связь, а также алгоритм Катхилла–Макки, позволяющий уменьшить пропускную способность разреженной матрицы расстояний. Для улучшения результатов отображения используются алгоритмы имитации отжига и алгоритм восхождения на вершину.

В работе [9] авторы представили два алгоритма, минимизирующих накладные расходы на коммуникации. Нисходящий алгоритм рекурсивно разбивает граф на уменьшающиеся блоки, восходящий идет в обратном порядке.

Известным инструментом является MPIPP, позволяющий осуществлять автоматический поиск оптимизированного отображения с целью минимизации затрат на связь типа точка-точка для приложений с произвольным характером обмена сообщениями [10].

В работе [11] предлагается метод отображения, учитывающий топологию коммуникационной среды с квадратичной временной сложностью. За счет применения метода на китайском суперкомпьютере Tianhe-2 авторам удалось сократить время выполнения реальных научных приложений (для случая 512 процессов) на 14,6 %.

В работе [12] реализован алгоритм параллельного блокового отображения. На каждый вычислительный узел поступает блок обработки подграфа информационного графа программы, содержащий связанные вершины графа. В работе [13] представлен параллельный алгоритм EagerMap, применяемый для по-

лучения отображения в кластерных системах. Алгоритм разбивает ветви параллельной программы на группы, в одну группу попадают наиболее связанные задачи. После объединения матрица смежности пересоздается для отображения связей уже не между задачами, а группами задач. Далее происходит непосредственное отображение групп на архитектуру системы.

Для поиска оптимального отображения в случаях больших систем и сложных приложений применяют эвристические алгоритмы, позволяющие находить приемлемое отображение за относительно короткое время. Эвристические алгоритмы можно разделить на итеративные и популяционные. К итеративным относятся алгоритмы поиска с запретами (поиск табу), жадного случайного адаптивного поиска, имитации отжига. К популяционным – генетические алгоритмы, алгоритм оптимизации роя частиц и муравьиный алгоритм.

В работе [14] сравниваются 11 эвристик, применяемых для поиска отображения. Наиболее точное решение дает применение генетического алгоритма. Алгоритм имитации отжига находит решение за самое короткое время, но при этом страдает точность отображения, поскольку на начальных этапах работы имитационный отжиг может принять очень плохие решения, от которых тяжело уйти на поздних этапах. Параллельная версия имитационного отжига рассмотрена в [15].

В [16] представлен сравнительный анализ использования алгоритма имитации отжига в задаче отображения. Приведены оптимальные функции перемещения во время генерации нового отображения, сравниваются функции принятия решения, описываются способы задания начальной и конечной температур и способ выбора количества итераций при заданных начальных параметрах алгоритма.

Авторы [17] отмечают, что при поиске отображения наилучший результат достигается путем применения генетического алгоритма, но время поиска отображения больше, чем у других алгоритмов.

В статье [18] сравниваются результаты поиска отображения при помощи генетического алгоритма и методов линейного программирования. В отличие от методов линейного программирования генетический алгоритм способен находить приближенное решение за приемлемое время. В работе [19] авторы применяют генетический алгоритм для решения задачи отображения на системах на кристалле.

Параллельная версия генетического алгоритма была предложена еще в [20]. В работах [21–23] авторы описывают наиболее часто используемую при реализации параллельного генетического алгоритма схему Master–Slave. Master отвечает за генерацию новых членов популяции. Все остальные вычислители Slave должны только рассчитывать значение целевой функции для новых членов популяции. В [21, 23] рассматривается параллельный генетический алгоритм с кольцевой схемой информационных обменов.

Получение графов программы и суперкомпьютерной системы

Входными данными для любого алгоритма поиска отображения являются программный граф и граф вычислительной системы. В системах коллективного пользования заданию динамически выделяется заранее неизвестное подмножество узлов (подсистема) суперкомпьютера. Получить граф выделенных узлов можно следующими способами:

– построение администратором системы полного графа всей суперкомпьютерной системы, из которого выделяется необходимый подграф;

– автоматизированное построение с использованием программных средств анализа топологии выделенного подмножества узлов (lb [24], netloc [25]);

– построение на основе данных систем мониторинга или утилит, предоставляемых вместе с коммуникационным оборудованием, которые позволяют по заданному списку вычислительных узлов получить граф выделенной подсистемы.

Первый способ построения графа подсистемы является достаточно трудоемким и не исключает появления ошибок при составлении графа вычислительной системы. Второй и третий способы обеспечивают необходимую точность и упрощают процесс построения графа. Для реализации третьего способа необходимо наличие в системе мониторинга функционала, совместимого с телекоммуникационным оборудованием суперкомпьютерного центра. Второй способ лишен указанных недостатков и может быть применен без каких-либо ограничений, однако требует регулярного запуска тестовых программ на всей вычислительной системе для составления и обновления сведений о ее структуре.

Для получения графа программы может быть использован один из следующих способов:

– предоставление графа пользователем;

– построение на основе данных программных средств анализа двоичного кода исполняемого файла и аппаратных счетчиков производительности (HPCToolkit [26], Intel Vtune Amplifire [27]);

– построение на основе данных инструментальных средств анализа программного кода (VampirTrace [28], MultiProcessing Environment [29], mpiP [30]).

Первый способ не требует использования специальных программных средств, однако он применим только для программ с регулярной структурой информационных обменов или невысокой степенью параллелизма. Второй и третий способы применимы для программ с любой степенью параллелизма. Второй способ преимущественно используется для анализа отдельных этапов параллельного алгоритма, поскольку предусматривает обработку статистической информации в интерактивном режиме. Программные средства, реализующие третий способ, предусматривают накопление данных о работе исследуемой программы в виде файлов трасс для их последующего анализа.

Параллельные алгоритмы поиска отображения

В данном исследовании для проведения сравнения авторы реализовали три эвристических параллельных алгоритма: имитации отжига, генетический и комбинированный алгоритмы.

Алгоритм имитации отжига основан на имитации физического процесса, происходящего во время отжига металлов. Суть алгоритма следующая.

1. Генерируется стартовое исходное решение, которое становится текущим.

2. Генерируется новое решение путем перестановки двух произвольных элементов матрицы X местами.

3. Если в результате перестановки приращение значения функционала (1) $\Delta F(X) < 0$, новое решение становится текущим. Если $\Delta F(X) > 0$, новое решение становится текущим с вероятностью, определяемой функцией принятия решения. Функция принятия решения зависит от текущей температуры отжига, понижающейся в процессе работы алгоритма.

4. Температура системы понижается в соответствии с видом функции понижения температуры.

5. Алгоритм останавливается по достижении одного из значений: определенного числа итераций, финальной температуры системы, числа подряд идущих итераций без улучшения значения целевой функции. Иначе – переход к п. 2.

В параллельной версии алгоритма несколько процессов (потоков) участвуют в поиске решения. Лучшее найденное решение сообщается всем процессам (потокам), после чего каждый из них выбирает полученное решение в качестве текущего. Параллельный имитационный отжиг позволяет за одно и то же время просмотреть большее количество вариантов отображения из пространства решений по сравнению с последовательной версией. Благодаря этому полученные решения покрывают большее количество локальных минимумов значений целевой функции и повышается вероятность нахождения решения, максимально близкого к глобальному минимуму целевой функции (1).

В терминах генетического алгоритма особь представляется массивом p, i-й элемент которого (ген) содержит номер узла, на который будет назначен i-й процесс параллельной программы. Особи составляют популяцию, размер которой равен или больше числа вершин программного графа. Операция скрещивания заключается в обмене генами между двумя особями популяции. Операция мутации с заданной вероятностью изменяет гены у заданного числа особей. Операция селекции выбирает из популяции особи с наименьшим значением функционала (1).

Для взаимодействия процессов, реализующих параллельный генетический алгоритм, могут быть применены две схемы информационного обмена. В первой схеме под названием Master–Slave выбирается главный процесс (Master), все остальные процессы являются подчиненными (Slave). Каждый подчиненный процесс на каждой итерации выполняет последовательный генетический алгоритм с заданным числом элементов популяции. После завершения последовательной итерации каждый процесс выбирает решение, имеющее минимальное значение целевой функции, и отправляет это решение главному процессу Master. Главный процесс производит генерацию новых членов популяции и, передавая эту информацию подчиненным процессам, запускает но-

вую итерацию алгоритма. Проведенные эксперименты показали невысокую эффективность схемы Master–Slave.

Альтернативой является кольцевая схема информационного обмена. На каждой итерации выполняющие алгоритм процессы обмениваются лучшими членами своих популяций. Количество передаваемых членов популяции не должно быть большим, так как это может привести к максимальной схожести популяций на начальных итерациях алгоритма. Каждый процесс в ходе алгоритма выполняет следующие шаги.

1. Генерация случайной начальной популяции и установка ее в качестве текущей.
2. Получение новых потомков применением операции скрещивания к выбранным особям текущей популяции.
3. Применение операции мутации к потомкам, полученным на предыдущем шаге.
4. Замена худших решений в текущей популяции новыми потомками.
5. Выбор на текущей итерации лучшего члена популяции.
6. Коммуникационный обмен лучшими решениями между соседними процессами.
7. Замена худшего решения в текущей популяции новым решением, если оно лучше.
8. Если не пройдено заданное число итераций, переход к п. 2.
9. Выбор особи с минимальным значением целевой функции (1) среди всех процессов. Выбранная особь принимается в качестве решения задачи отображения.

Сгенерированные решения в генетических алгоритмах за счет операторов скрещивания и мутации позволяют охватить больше локальных минимумов целевой функции и максимально приблизиться к значению глобального минимума. Преимуществом алгоритма имитации отжига является меньшее время работы поиска приемлемого решения. Продуктивной видится идея комбинирования двух этих алгоритмов, чтобы обеспечить высокую точность отображения за приемлемое время. Рассмотрим предлагаемый комбинированный параллельный алгоритм.

На первом этапе комбинированного алгоритма работает параллельный алгоритм имитации отжига. В отличие от рассмотренного выше алгоритма имитации отжига данный алгоритм не подразумевает обмена между процессами. Каждый процесс генерирует заданное количество решений, которые для параллельного генетического алгоритма становятся текущей популяцией. Выбор типов операторов генетического алгоритма осуществляется в соответствии с заданной конфигурацией.

Параллельный комбинированный алгоритм поиска отображения состоит из следующих шагов.

1. Параллельный поиск решений каждым процессом с применением имитационного отжига.
2. Создание популяции решений для работы параллельного генетического алгоритма из сгенерированных решений на шаге 1.
3. Запуск параллельного генетического алгоритма на заданное число итераций.
4. Выбор лучшего решения в каждом процессе.
5. Выбор лучшего глобального решения.

Отсутствие обмена на этапе работы параллельного алгоритма имитации отжига гарантирует, что каждый процесс сгенерирует уникальную популяцию решений. За счет миграции решений удается передавать между популяциями лучшие признаки, которые будут наследованы потомками.

Библиотека UGR-Metaheuristics и ее дополнение

Для реализации рассмотренных алгоритмов применена свободно распространяемая библиотека UGR-Metaheuristics [31], дополненная разработанным комбинированным алгоритмом. Библиотека UGR-Metaheuristics написана на языке программирования C++ и реализует большинство известных эвристических алгоритмов решения задачи отображения.

Базовым классом библиотеки является класс Solver, конструктор которого принимает матрицы расстояний графа вычислительной системы и программного графа. Класс Solution представляет собой универсальное решение, которое каждый алгоритм получает во время своей работы. В этом классе в виде вектора хранится текущее отображение, генерируется произвольное отображение, происходит оптимизированный расчет значения целевой функции.

Рассмотрим дополнения к библиотеке UGR-Metaheuristics, показанные на рисунке 2. Для параллельного алгоритма имитации отжига образован класс ParallelSA. Одним из членов класса является массив, состоящий из экземпляров класса simAnnealing (базовый класс библиотеки). Количество элементов в массиве задается пользователем с помощью настроек.

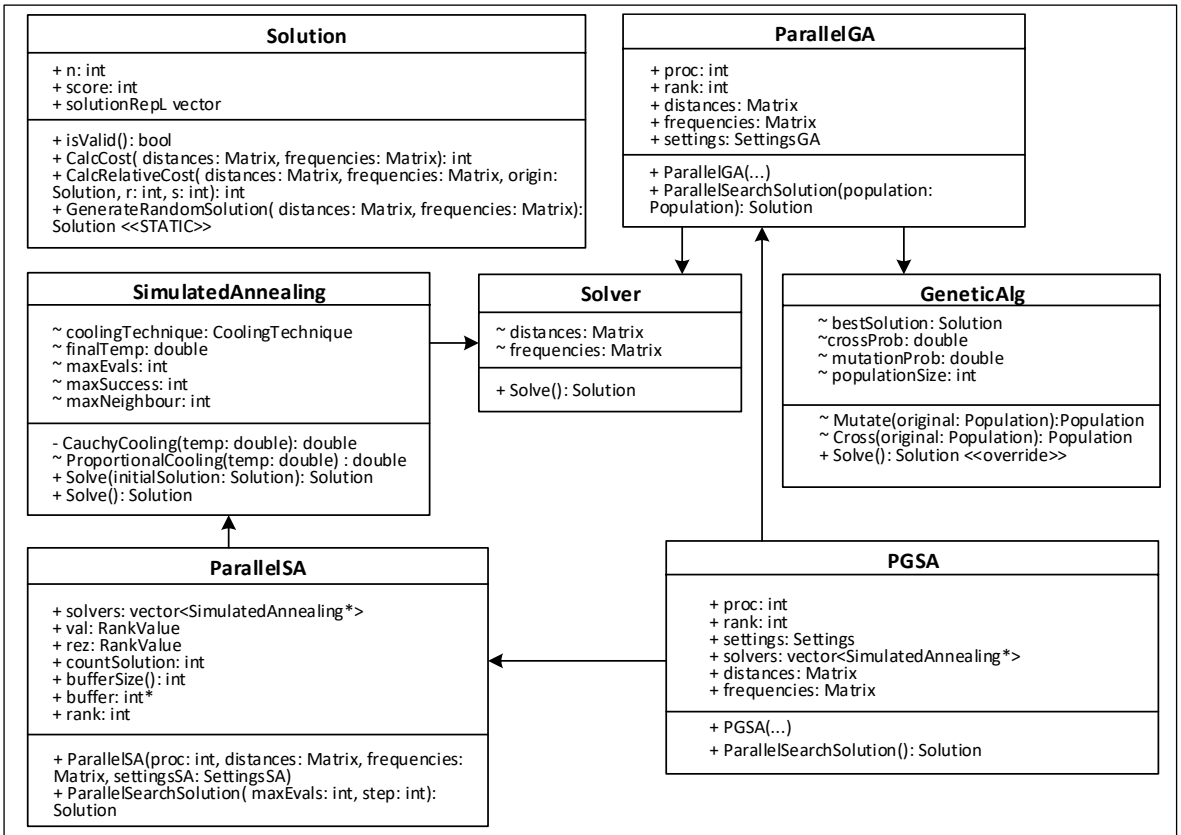

Рис. 2. Дополнения к библиотеке UGR-Metaheuristics

Fig. 2. Add-ons for the UGR-Metaheuristics library

Внутри класса ParallelSA сохраняется лучшее решение, полученное при помощи последовательного алгоритма имитации отжига. Ключевым методом класса является ParallelSA::ParallelSearchSolution, в цикле которого на каждой итерации находится лучшее локальное решение. Далее с помощью редукции осуществляется поиск лучшего глобального решения среди параллельных процессов, выполняющих алгоритм. Найденное лучшее глобальное решение становится текущим для каждого процесса, и на следующей итерации поиск будет осуществлен относительно лучшего решения.

Параллельный генетический алгоритм в библиотеке UGR-Metaheuristics реализован базовым классом GeneticAlg, в который были добавлены два метода: CircleSolve и MasterSlaveSolve. Методы вызываются в каждом процессе, реализующем параллельный генетический алгоритм, и используют метод GeneticAlg::Solve, который в каждом процессе ищет локальное решение задачи отображения при помощи последовательного генетического алгоритма. Метод MasterSlaveSolve реализует схему Master–Slave и после вызова метода GeneticAlg::Solve с помощью операции редукции среди всех локальных находит лучшее глобальное решение.

Метод CircleSolve реализует кольцевую схему информационного обмена. Создается начальная популяция, которая становится текущей. Внутри цикла к текущей популяции применяются операторы скрещивания и мутации, затем в текущей популяции заменяются худшие члены. В каждом процессе выбирается лучшее локальное решение, после чего осуществляется кольцевой обмен. После обмена происходит замена худших членов и новая популяция становится текущей на следующей итерации алгоритма. После того как алгоритм прошел заданное число итераций, с помощью операций редукции выбирается лучшее глобальное решение.

Комбинированный алгоритм реализован классом PGSA, который сначала в каждом процессе запускает последовательный алгоритм имитации отжига. Результатом является начальная популяция, с которой запускается параллельный генетический алгоритм с кольцевой или Master–Slave схемой обмена.

Программное средство построения отображения GraphHunter

В основу разработанного программного средства GraphHunter, решающего задачу построения отображения, легли технологии MPI и OpenMP, а также библиотека поиска отображения UGR-Metaheuristics и библиотека program_options пакета boost, с помощью которой можно осуществлять взаимодействие с файлами настроек и аргументами командной строки.

На рисунке 3 представлена схема разработанной программы.

В состав программного средства входят четыре основных модуля.

1. Модуль загрузки настроек программы, в котором происходят чтение и обработка конфигурационного файла. К основным настройкам относятся выбранный алгоритм отображения, параметры библиотеки UGR-Metaheuristics [31] для параллельных генетического алгоритма и алгоритма имитации отжига, параметры разработанного комбинированного алгоритма.

2. Модуль построения матрицы расстояний графа вычислительной системы позволяет автоматически определять коммуникационный граф выделенной параллельной программе подсистемы суперкомпьютера. Напомним, подсистема выделяется системой коллективного пользования после прохождения параллельной программы в составе задания через очередь. Для построения матрицы расстояний используются метод и программные средства, определяющие латентность каналов связи между выделенными параллельной программе узлами суперкомпьютера [5].

3. Модуль построения матрицы расстояний программного графа. Для построения матрицы применяется анализ файлов трассировки – журналов событий, произошедших во время выполнения программы. Предполагается, что пользователь предоставит файлы трассы своей программы, для чего воспользуется специализированной программой, например Vampir-Trace [28]. Файлы трассировки содержат число и размер сообщений, передаваемых между узлами. Элементом (i, j) матрицы расстояний будет время передачи сообщений $t_{i,j}$ между соответствующими узлами, которое рассчитывается по формуле $t_{i,j} = L \times N_{i,j} + D_{i,j}/S$, где L – латентность сети; $N_{i,j}$ – число передаваемых сообщений между узлами i и j; $D_{i,j}$ – общий размер передаваемых сообщений; S – пропускная способность сети.

В исходный код программного средства igtrace, осуществляющего чтение файлов трассировки, была добавлена функция tracestat_get_taskgraph, результатом работы которой является построенная матрица расстояний программного графа.

4. Модуль построения отображения. При запуске разработанного программного средства будет использоваться указанный в конфигурационном файле один из трех рассмотренных выше параллельных алгоритмов: имитации отжига, генетический или комбинированный.

К средствам обработки выходных данных следует отнести модуль создания файла отображения. Результатом его работы является текстовый файл в стандарте machinefile, поддер-

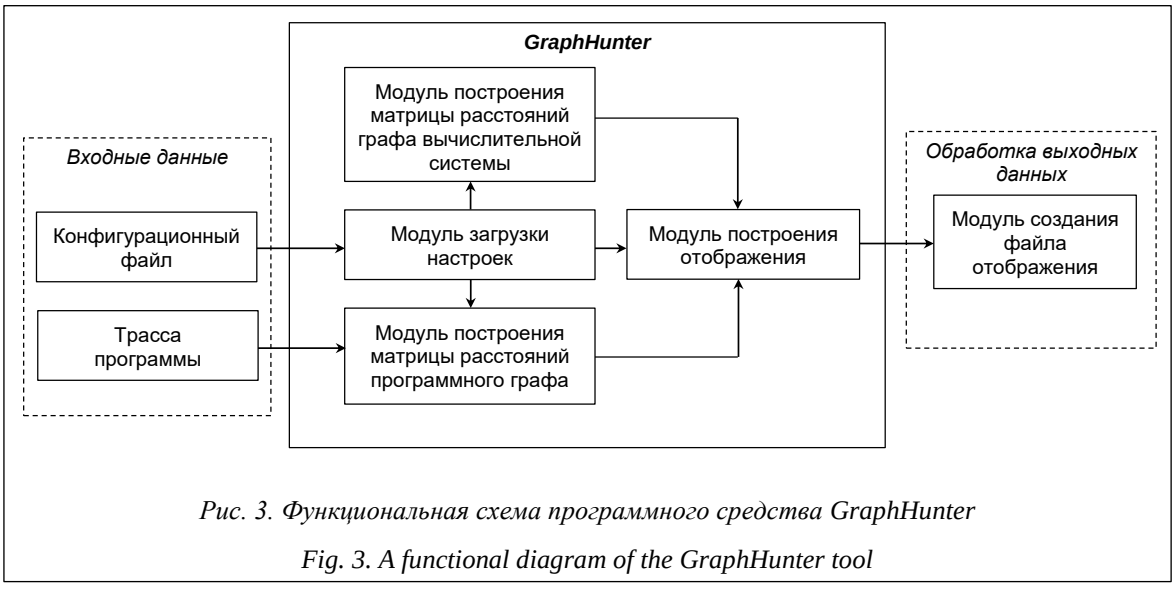

Рис. 3. Функциональная схема программного средства GraphHunter

Fig. 3. A functional diagram of the GraphHunter tool

живаемом большинством реализаций MPI [32]. Файл задает распределение MPI-процессов по выделенным заданию вычислительным узлам.

Оценка точности отображения A_1 рассчитывается как

$$A_1 = 100(f_1 - f_0)/f_0, \quad (2)$$

где f_1 – значение целевой функции для построенного отображения; f_0 – значение целевой функции оптимального отображения. Оценка A_1 показывает отклонение значения целевой функции построенного отображения от оптимального значения.

Методика проведения и результаты экспериментов

Программу GraphHunter экспериментально запускали на разделе Broadwell суперкомпьютера МВС-10П ОП [33], установленного в МСЦ РАН. Раздел состоит из 136 двухпроцессорных узлов на базе 16-ядерных процессоров Intel Xeon E5-2697Av4.

В качестве входных данных использовался набор taiXeyy [34, 35] графов различных порядков, где X – порядок графов, yy – вариант графов заданного порядка. В наборе для каждого порядка указаны граф вычислительной системы и информационный граф программы, представленные в виде матриц расстояний. Для каждой пары графов известны матрицы расстояний и минимальное значение целевой функции (1). Набор taiXeyy используется учеными для тестирования алгоритмов отображения при решении задач о квадратичном назначении, частным случаем которых является задача отображения.

В экспериментах использовались наборы графов tai27e01, tai45e01, tai75e01, tai125e01, tai175e01, tai343e01, tai729e01. Каждый эксперимент состоял в поиске отображения для заданной пары графов с фиксированными параметрами алгоритмов и числом процессов, участвующих в поиске отображения. Для усреднения результатов для каждого эксперимента было проведено по 10 запусков с одними и теми же параметрами.

На первом этапе были определены оптимальные параметры алгоритмов, обеспечивающие наиболее точные решения. Подбор параметров осуществлялся в формате проведения вычислительного эксперимента. Эксперимент заключался в многократном поиске отображения для графов заданного порядка с разным числом ветвей параллельной программы с фиксированными параметрами, за исключением подбираемого параметра – его значения варьировались.

Приведем пример подбора параметра. В рамках одного значения температуры алгоритм имитации отжига просматривает несколько потенциально возможных решений. При возрастании числа просматриваемых решений при одном значении температуры увеличивается время поиска отображения. Эксперименты показали, что лучшее среднее значение целевой функции достигается при числе просматриваемых решений, равном 50. При этом алгоритм имеет приемлемое среднее время поиска отображения.

Другим важным параметром является функция понижения температуры. Рассмотрены две реализованные в библиотеке UGR-Metaheuristics функции: пропорциональная и функция Коши. В экспериментах функция Коши показала минимальное среднее время выполнения и минимальное среднее значение целевой функции.

Параллельный алгоритм заключается в одновременной работе нескольких процессов, каждый из которых выполняет некоторое число последовательных итераций алгоритма имитации отжига. По истечении этого числа итераций процессы обмениваются лучшими найденными решениями. Для набора tai343e01 лучшее качество решения достигается при прохождении 100 последовательных итераций на один информационный обмен. На графах других порядков среднее лучшее решение достигается при 1 000 последовательных итераций на один информационный обмен.

Общее число итераций (N) параллельного алгоритма связано с числом последовательных итераций по следующей формуле: $N = cn$, где c – количество информационных обменов; n – число последовательных итераций. При фиксированном числе итераций параллельного алгоритма уменьшение числа последовательных итераций приводит к увеличению обменов. Число обменов влияет на время выполнения алгоритма.

В зависимости от порядка графа может понадобиться различное число итераций параллельного алгоритма. В ходе подбора значений параметра было выявлено:

– для графов порядка не более 256 достаточно 50 000 итераций;

– для графов порядка не более 1 024 достаточно 100 000 итераций.

Оценка точности полученных решений

Accuracy estimation of the obtained solution

Набор данных	Алгоритм										
	Параллельный имитационный отжиг			Параллельный генетический			Параллельный комбинированный			Оптимальное решение	
	F	T	A_1	F	T	A_1	F	T	A_1	F_O	T_O
Tai27	2558	0,05	1	3176	0,1	24	2600	0,27	2	2558	0,02
Tai45	6724	0,3	5	8564	0,45	34	7332	0,5	14	6412	0,03
Tai75	19380	0,6	34	18268	0,7	26	18810	0,75	29	14488	8
Tai125	50780	1,6	43	47816	2	35	50792	1,75	43	35426	166
Tai175	72688	2,8	26	74602	5	29	74880	3,1	29	57540	181
Tai343	200856	3,5	37	168120	12,8	15	172466	10,1	18	145862	1026
Tai729	724820	18,2	54	514846	50	9	498454	53,2	6	469650	1187

При выполнении имитационного отжига в каждом процессе одновременно работают несколько решателей, которые пытаются улучшить текущее решение. Эксперименты показали, что для графов порядка не более 100 число решателей совпадает с порядком графа. Для графов порядка до 1 024 количество решателей должно составлять 125.

Для параллельного генетического алгоритма были применены параметры, рекомендуемые автором библиотеки UGR-Metaheuristics, за исключением размера популяции.

На втором этапе алгоритмы сравнивались по точности, определяемой в соответствии с (2), и времени выполнения. Параллельная схема алгоритмов такова, что увеличение числа процессов приводит к расширению пространства рассматриваемых решений. Соответственно, изменение числа процессов влияет на точность отображения, практически не сказываясь на времени работы алгоритмов.

Эксперименты показали, что генетический алгоритм при поиске отображения больших графов в среднем получает решение лучше, чем имитационный отжиг, но при этом на графах больших порядков имеет существенно большее время выполнения, неприемлемое для динамического поиска отображения в системах коллективного пользования.

В таблице содержатся оценки точности и времени выполнения алгоритмов по итогам проведенных экспериментов. Оптимальные решения для наборов данных были взяты из http://mistic.heig-vd.ch/taillard/problemes.dir/qap.dir/summary_bvk.txt. В столбце F указаны полученные в ходе экспериментов лучшие значения целевой функции, в T – время в минутах, затраченное на получение лучшего решения. В столбце F_O указаны оптимальные значения целевой функции для каждого из наборов, в T_O – время получения оптимального отображения из работы [36]. В столбце A_1 приведены оценки точности полученного решения в соответствии с (2).

На третьем этапе была произведена оценка масштабируемости программного средства GrpahHunter. Особенность рассматриваемых алгоритмов в том, что увеличение числа реализующих алгоритм процессов до определенных пределов приводит к получению более точного решения. Для набора tai343e01 зависимость точности отображения от числа процессов программы GraphHunter показана на рисунке 4. Нетрудно заметить рост точности с увеличением числа процессов.

Для 256 процессов определена масштабируемость программы GraphHunter при запуске ее для набора tai343e01 на разном числе узлов раздела Broadwell суперкомпьютера МВС-10П

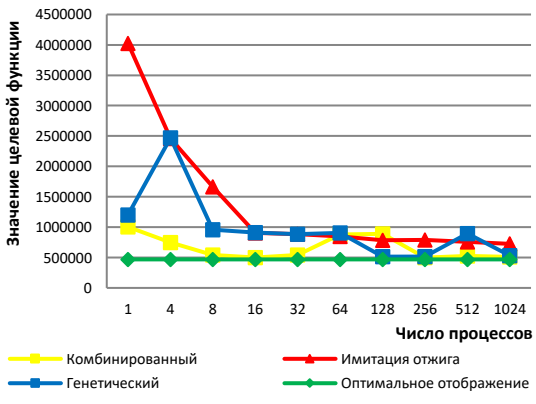

Рис. 4. Точность отображения для набора tai343e01 в зависимости от числа процессов программного средства GraphHunter

Fig. 4. The mapping accuracy for the tai343e01 instance depending on the number of GraphHunter processes

ОП. Представленные на рисунке 5 результаты позволяют сделать вывод о достаточно хорошей масштабируемости разработанного приложения.

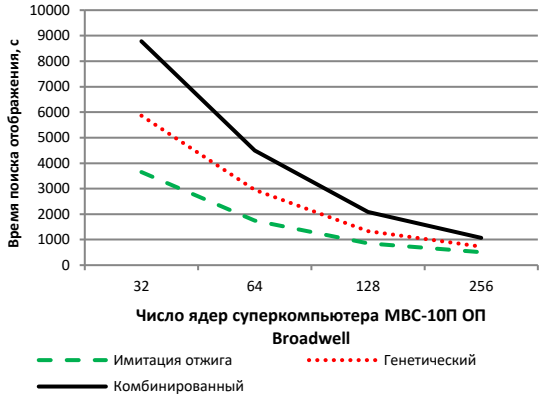

Рис. 5. Масштабируемость программного средства GraphHunter при поиске отображения 256 процессами для набора tai343e01

Fig. 5. The GraphHunter tool scalability while mapping by 256 processes for the tai343e01 instance

Заключение

Задачу оптимального размещения процессов параллельной программы на узлах суперкомпьютерных систем можно решить с помощью большого количества методов и алгоритмов. Для использования в системе управления заданиями суперкомпьютера МВС-10П ОП были реализованы параллельные алгоритмы поиска отображения программного графа на граф суперкомпьютерной системы с различными характеристиками. Исследовать характеристики этих алгоритмов позволяет разработанное программное средство GraphHunter. Программа GraphHunter базируется на библиотеке алгоритмов отображения UGR-Metaheuristics, которая была дополнена классами, реализующими параллельные алгоритмы: имитационного отжига, генетический и комбинированный алгоритмы. Экспериментальные исследования программы GraphHunter произведены на суперкомпьютере МВС-10П ОП (раздел Broadwell). По результатам экспериментов можно сделать следующие выводы:

– GraphHunter позволяет подобрать оптимальные параметры реализованных алгоритмов, обеспечивающие высокую точность отображения при приемлемом для систем коллективного пользования времени выполнения;

– GraphHunter позволяет реализовать и сравнить по точности и времени выполнения произвольный параллельный алгоритм поиска отображения, в частности, в ходе экспериментов получены оценки для реализованных алгоритмов имитации отжига, генетического и их комбинации;

– точность отображения зависит от числа процессов программы GraphHunter, выполняющих алгоритм;

– GraphHunter показал близкую к линейной масштабируемость при заданной точности отображения.

В дальнейшем GraphHunter может быть применена в системах управления заданиями для реализации двухэтапного метода РГО, который предполагает запуск алгоритмов отображения на выделенных для очередного задания узлах суперкомпьютера.

Работа выполнена в МСЦ РАН в рамках государственного задания по теме FNEF-2022-0016.

Литература

1. Монахов О.Г., Гросбейн Е.Б., Мусин А.Р. Алгоритмы отображения для параллельных систем на основе моделирования эволюции и моделирования отжига // Распределенная обработка информации: тр. VI Междунар. семинара. 1998. С. 142–146.

2. Баранов А.В. Метод и алгоритмы осуществления оптимального отображения параллельной программы на структуру многопроцессорного вычислителя // Высокопроизводительные вычисления и их приложения: матер. конф. 2000. С. 65–67.

3. Забродин А.В., Левин В.К., Корнеев В.В. Массово параллельные системы МВС-100 и МВС-1000 // Научная сессия МИФИ. 2000. Т. 2. С. 194–195.

4. Фещенко В.П., Матюшков Л.П. Итерационный алгоритм разрезания графа на К подграфов // Автоматизация проектирования сложных систем. 1976. № 2. С. 74–77.

5. Киселёв Е.А., Корнеев В.В. Оптимизация отображения программ на вычислительные ресурсы // Программная инженерия. 2015. № 8. С. 3–8.

6. Deveci M., Kaya K., Uçar B., Çatalyürek Ü.V. Fast and high quality topology-aware task mapping. Proc. IEEE Int. Parallel and Distributed Processing Symposium, 2015, pp. 197–206. DOI: 10.1109/IPDPS.2015.93.

7. Pellegrini F. Scotch and PT-scotch graph partitioning software: An overview. In: Combinatorial Scientific Computing, 2012, pp. 373–406. DOI: 10.1201/B11644-15.

8. Hoefler T., Snir M. Generic topology mapping strategies for large-scale parallel architectures. Proc. ICS, 2011, pp. 75–84. DOI: 10.1145/1995896.1995909.

9. Kirchbach K.V., Schulz C., Träff J.L. Better process mapping and sparse quadratic assignment. ACM J. of Experimental Algorithmics, 2020, vol. 25, art. 1.11, pp. 1–19. DOI: 10.1145/3409667.

10. Chen H., Chen W., Huang J., Robert B., Kuhn H. MPIPP: an automatic profile-guided parallel process placement toolset for SMP clusters and multiclusters. Proc. XX ICS, 2006, pp. 353–360. DOI: 10.1145/1183401.1183451.

11. Yan B., Xiao L., Qin G. et al. QTMS: A quadratic time complexity topology-aware process mapping method for large-scale parallel applications on shared HPC system. Parallel Computing, 2020, vol. 94-95, art. 102637. DOI: 10.1016/j.parco.2020.102637.

12. Predari M., Tzovas C., Schulz C., Meyerhenke H. An MPI-based algorithm for mapping complex networks onto hierarchical architectures. Proc. Euro-Par. Lecture Notes in Computer Science, 2021, vol. 12820, pp. 167–182. DOI: 10.1007/978-3-030-85665-6_11.

13. Cruz E., Diener M., Pilla L., Navaux P. EagerMap: A task mapping algorithm to improve communication and load balancing in clusters of multicore systems. ACM Transactions on Parallel Computing, 2019, vol. 5, no. 4, art. 17. DOI: 10.1145/3309711.

14. Braun T.D., Siegel H.J., Beck N., Bölöni L.L. et al. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. J. of Parallel and Distributed Computing, 2001, vol. 61, no. 6, pp. 810–837. DOI: 10.1006/jpdc.2000.1714.

15. Galea F., Sirdey R. A parallel simulated annealing approach for the mapping of large process networks. Proc. IEEE XXVI Int. Parallel and Distributed Processing Symposium Workshops & PhD Forum, 2012, pp. 1787–1792. DOI: 10.1109/IPDPSW.2012.221.

16. Orsila H., Salminen E., Hamalainen T. Recommendations for using simulated annealing in task mapping. Design Automation for Embedded Systems, 2013, vol. 17, pp. 53–85. DOI: 10.1007/s10617-013-9119-0.

17. Gupta M., Bhargava L., Indu S. Mapping techniques in multicore processors: Current and future trends. The J. of Supercomputing, 2021, vol. 77, pp. 9308–9363. DOI: 10.1007/s11227-021-03650-6.

18. Azketa E., Uribe J.P., Marcos M., Almeida L., Gutierrez J.J. Permutational genetic algorithm for the optimized assignment of priorities to tasks and messages in distributed real-time systems. Proc. X Int. Conf. on Trust, Security and Privacy in Computing and Communications, 2011, pp. 958–965. DOI: 10.1109/TrustCom.2011.132.

19. de Rocha A., Beck A., Maia S., Kreutz M., Pereira M. A routing based genetic algorithm for task mapping on MPSoC. Proc. X SBESC, 2020, pp. 1–8. DOI: 10.1109/SBESC51047.2020.9277843.

20. Talbi E.-G., Bessiere P. A parallel genetic algorithm for the graph partitioning problem. Proc. V ICS, 1991, pp. 312–320. DOI: 10.1145/109025.109102.

21. Rivera W. Scalable parallel genetic algorithms. Artificial Intelligence Review, 2001, vol. 16, pp. 153–168.

22. Millan J., Calvo V., Chaves R. Quadratic assignment problem (QAP) on GPU through a master-slave PGA. Vision Electronica, 2016, vol. 10, no. 2, pp. 179–183. DOI: 10.14483/22484728.11738.

23. Lakhdar L., Mehdi M., Melab N., Talbi E-G. Parallel hybrid genetic algorithms for solving Q3AP on computational grid. Int. J. of Foundations of Computer Science, 2012, vol. 23, no. 2, pp. 483–500. DOI: 10.1142/S0129054112400242.

24. Аладышев О.С., Киселев Е.А., Корнеев В.В., Шабанов Б.М. Программные средства построения коммуникационной схемы многопроцессорной вычислительной системы // СКТ-2010: матер. Междунар. науч.-технич. конф. 2010. Т. 1. С. 176–179.

25. Goglin B., Hursey J., Squyres J. Netloc: Towards a comprehensive view of the HPC system topology. Proc. XLIII Int. Conf. on Parallel Processing Workshops, 2014, pp. 216–225. DOI: 10.1109/ICPPW.2014.38.

26. Meng X., Anderson J.M., Mellor-Crummey J. et al. Parallel binary code analysis. Proc. XXVI ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, 2021, pp. 76–89. DOI: 10.1145/3437801.3441604.

27. Pandey A., Tesfay D., Jarso E. Performance analysis of Intel Ivy Bridge and Intel Broadwell microarchitectures using Intel VTune amplifier software. Proc. II ICISC, 2018, pp. 423–426. DOI: 10.1109/ICISC.2018.8399107.

28. Brunst H., Knüpfer A. Vampir. In: Encyclopedia of Parallel Computing. Boston, Springer Publ., 2011, pp. 2125–2129. DOI: 10.1007/978-0-387-09766-4_60.

29. Chan A., Gropp W., Lusk E. An efficient format for nearly constant-time access to arbitrary time intervals in large trace files. Scientific Programming, 2008, vol. 16, no. 2-3, pp. 155–165.

30. Roth P., Meredith J., Vetter J. Automated characterization of parallel application communication patterns. Proc. XXIV Int. Symposium on High-Performance Parallel and Distributed Computing, 2015, pp. 73–84. DOI: 10.1145/2749246.2749278.

31. Metaheuristica – Practica 3.a. URL: https://raw.githubusercontent.com/salvacorts/UGR-Metaheuristics/P3/doc/memoria/memoria.pdf (дата обращения: 03.07.2022).

32. Intel® MPI Library Reference Manual. URL: https://www.lrz.de/services/compute/linux-cluster/documentation/protected_nicht_fiona/mpi_30/Reference_Manual.pdf (дата обращения: 03.07.2022).

33. Savin G.I., Shabanov B.M., Telegin P.N., Baranov A.V. Joint supercomputer center of the Russian academy of sciences: Present and future. Lobachevskii J. of Mathematics, 2019, vol. 40, pp. 1853–1862. DOI: 10.1134/S1995080219110271.

34. Drezner Z., Hahn P., Taillard E. Recent Advances for the quadratic assignment problem with special emphasis on instances that are difficult for meta-heuristic methods. Annals of Operations Research, 2005, vol. 139, pp. 65–94. DOI: 10.1007/s10479-005-3444-z.

35. Quadratic Assignment Instances. URL: http://mistic.heig-vd.ch/taillard/problemes.dir/qap.dir/qap.html (дата обращения: 03.07.2022).

36. Mihic K., Ryan K., Wood A. Randomized decomposition solver with the quadratic assignment problem as a case study. INFORMS J. on Computing, 2018, vol. 30, no. 2, pp. 295–308. DOI: 10.1287/ijoc.2017.0781.

Software & Systems
DOI: 10.15827/0236-235X.140.583-597

Received 31.08.22, Revised 14.09.22
2022, vol. 35, no. 4, pp. 583–597

A GraphHunter software tool for mapping parallel programs to a supercomputer system structure

A.V. Baranov [1], *Ph.D. (Engineering), Associate Professor, Leading Researcher, antbar@mail.ru, abaranov@jscc.ru*
E.A. Kiselev [1], *Ph.D. (Engineering), Senior Researcher, e.a.kiselev@yandex.ru, kiselev@jscc.ru*
P.N. Telegin [1], *Ph.D. (Engineering), Leading Researcher, pnt@jscc.ru*
A.A. Sorokin [2], *Student, rexantmaster@yandex.ru*

[1] *Joint Supercomputer Center of the RAS – branch of Federal State Institution "Scientific Research Institute for System Analysis of the RAS", Moscow, 119334, Russian Federation*
[2] *MIREA – Russian Technological University, Moscow, 119454, Russian Federation*

Abstract. One of well-known problems in high-performance computing is optimal mapping of parallel program processes to supercomputer system nodes. A solution for this problem minimizes the overhead for information exchanges between the processes of a parallel program and thus increases the performance of calculations. When solving a mapping problem, both a supercomputer system and a parallel program are represented as graphs.

The paper shows solving the mapping problem in relation to a system for collective use of a supercomputer that handles a queue of parallel programs. After passing the queue, a new previously unknown subset of supercomputer nodes is allocated to the parallel program. In this case, it is necessary to construct a graph of a selected subset of nodes and find a suitable mapping of the parallel program onto this graph in a reasonable time. It is suggested to run parallel mapping algorithms on the supercomputer nodes allocated for parallel program.

To study the properties of mapping algorithms, the GraphHunter software tool was developed. This tool makes it possible to conduct experiments with three parallel algorithms: simulated annealing, genetic algorithm, and their combination. This article discusses the structure of the GraphHunter software tool, and presents the results of experiments with GraphHunter runs on the MVS-10P OP supercomputer at the Joint Supercomputing Center of the Russian Academy of Sciences.

Keywords: high performance computing, parallel mapping algorithm, simulated annealing, genetic algorithm, job scheduling.

Acknowledgements. *The work has been carried out at the JSC RAS within the framework of the state task on the topic FNEF-2022-0016.*

References

1. Monakhov O.G., Grosbeyn E.B., Musin A.R. Mapping algorithms for parallel systems based on evolution simulation and simulated annealing. *Proc. VI Int. Workshop Distributed Data Processing*, 1998, pp. 142–146 (in Russ.).
2. Baranov A.V. Method and algorithms for a parallel program optimal mapping to the multiprocessor computer structure. *Proc. Conf. High Performance Computing and its Applications*, 2000, pp. 65–67 (in Russ.).
3. Zabrodin A.V., Levin V.K., Korneev V.V. Massively parallel systems MVS-100 and MVS-1000. *Scientific Session of MEPI*, 2000, vol. 2, pp. 194–195 (in Russ.).
4. Feschenko V.P., Matyushkov L.P. An iterative algorithm for partitioning a graph into K subgraphs. *Automation of the complex Systems Design*, 1976, vol. 2, pp. 74–77 (in Russ.).
5. Kiselev E.A., Korneev V.V. Optimization of mapping programs to computing resources. *Software Engineering*, 2015, no 8, pp. 3–8 (in Russ.).
6. Deveci M., Kaya K., Uçar B., Çatalyürek Ü.V. Fast and high quality topology-aware task mapping. *Proc. IEEE Int. Parallel and Distributed Processing Symposium*, 2015, pp. 197–206. DOI: 10.1109/IPDPS.2015.93.
7. Pellegrini F. Scotch and PT-scotch graph partitioning software: An overview. In: *Combinatorial Scientific Computing*, 2012, pp. 373–406. DOI: 10.1201/B11644-15.
8. Hoefler T., Snir M. Generic topology mapping strategies for large-scale parallel architectures. *Proc. ICS*, 2011, pp. 75–84. DOI: 10.1145/1995896.1995909.
9. Kirchbach K.V., Schulz C., Träff J.L. Better process mapping and sparse quadratic assignment. *ACM J. of Experimental Algorithmics*, 2020, vol. 25, art. 1.11, pp. 1–19. DOI: 10.1145/3409667.
10. Chen H., Chen W., Huang J., Robert B., Kuhn H. MPIPP: an automatic profile-guided parallel process placement toolset for SMP clusters and multiclusters. *Proc. XX ICS*, 2006, pp. 353–360. DOI: 10.1145/1183401.1183451.
11. Yan B., Xiao L., Qin G. et al. QTMS: A quadratic time complexity topology-aware process mapping method for large-scale parallel applications on shared HPC system. *Parallel Computing*, 2020, vol. 94-95, art. 102637. DOI: 10.1016/j.parco.2020.102637.
12. Predari M., Tzovas C., Schulz C., Meyerhenke H. An MPI-based algorithm for mapping complex networks onto hierarchical architectures. *Proc. Euro-Par. Lecture Notes in Computer Science*, 2021, vol. 12820, pp. 167–182. DOI: 10.1007/978-3-030-85665-6_11.
13. Cruz E., Diener M., Pilla L., Navaux P. EagerMap: A task mapping algorithm to improve communication and load balancing in clusters of multicore systems. *ACM Transactions on Parallel Computing*, 2019, vol. 5, no. 4, art. 17. DOI: 10.1145/3309711.
14. Braun T.D., Siegel H.J., Beck N., Bölöni L.L. et al. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *J. of Parallel and Distributed Computing*, 2001, vol. 61, no. 6, pp. 810–837. DOI: 10.1006/jpdc.2000.1714.
15. Galea F., Sirdey R. A parallel simulated annealing approach for the mapping of large process networks. *Proc. IEEE XXVI Int. Parallel and Distributed Processing Symposium Workshops & PhD Forum*, 2012, pp. 1787–1792. DOI: 10.1109/IPDPSW.2012.221.
16. Orsila H., Salminen E., Hamalainen T. Recommendations for using simulated annealing in task mapping. *Design Automation for Embedded Systems*, 2013, vol. 17, pp. 53–85. DOI: 10.1007/s10617-013-9119-0.
17. Gupta M., Bhargava L., Indu S. Mapping techniques in multicore processors: Current and future trends. *The J. of Supercomputing*, 2021, vol. 77, pp. 9308–9363. DOI: 10.1007/s11227-021-03650-6.
18. Azketa E., Uribe J.P., Marcos M., Almeida L., Gutierrez J.J. Permutational genetic algorithm for the optimized assignment of priorities to tasks and messages in distributed real-time systems. *Proc. X Int. Conf. on Trust, Security and Privacy in Computing and Communications*, 2011, pp. 958–965. DOI: 10.1109/TrustCom.2011.132.
19. de Rocha A., Beck A., Maia S., Kreutz M., Pereira M. A routing based genetic algorithm for task mapping on MPSoC. *Proc. X SBESC*, 2020, pp. 1–8. DOI: 10.1109/SBESC51047.2020.9277843.
20. Talbi E.-G., Bessiere P. A parallel genetic algorithm for the graph partitioning problem. *Proc. V ICS*, 1991, pp. 312–320. DOI: 10.1145/109025.109102.
21. Rivera W. Scalable parallel genetic algorithms. *Artificial Intelligence Review*, 2001, vol. 16, pp. 153–168.
22. Millan J., Calvo V., Chaves R. Quadratic assignment problem (QAP) on GPU through a master-slave PGA. *Vision Electronica*, 2016, vol. 10, no. 2, pp. 179–183. DOI: 10.14483/22484728.11738.
23. Lakhdar L., Mehdi M., Melab N., Talbi E-G. Parallel hybrid genetic algorithms for solving Q3AP on computational grid. *Int. J. of Foundations of Computer Science*, 2012, vol. 23, no. 2, pp. 483–500. DOI: 10.1142/S0129054112400242.

24. Aladyshev O.S., Kiselev E.A., Korneev V.V., Shabanov B.M. Software tools for constructing a communication scheme of a multiprocessor computer. *Proc. SCT-2010*, 2010, vol. 1, pp. 176–179 (in Russ.).

25. Goglin B., Hursey J., Squyres J. Netloc: Towards a comprehensive view of the HPC system topology. *Proc. XLIII Int. Conf. on Parallel Processing Workshops*, 2014, pp. 216–225. DOI: 10.1109/ICPPW.2014.38.

26. Meng X., Anderson J.M., Mellor-Crummey J. et al. Parallel binary code analysis. *Proc. XXVI ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2021, pp. 76–89. DOI: 10.1145/3437801.3441604.

27. Pandey A., Tesfay D., Jarso E. Performance analysis of Intel Ivy Bridge and Intel Broadwell microarchitectures using Intel VTune amplifier software. *Proc. II ICISC*, 2018, pp. 423–426. DOI: 10.1109/ICISC.2018.8399107.

28. Brunst H., Knüpfer A. Vampir. In: *Encyclopedia of Parallel Computing*. Boston, Springer Publ., 2011, pp. 2125–2129. DOI: 10.1007/978-0-387-09766-4_60.

29. Chan A., Gropp W., Lusk E. An efficient format for nearly constant-time access to arbitrary time intervals in large trace files. *Scientific Programming*, 2008, vol. 16, no. 2-3, pp. 155–165.

30. Roth P., Meredith J., Vetter J. Automated characterization of parallel application communication patterns. *Proc. XXIV Int. Symposium on High-Performance Parallel and Distributed Computing*, 2015, pp. 73–84. DOI: 10.1145/2749246.2749278.

31. *Metaheuristica – Practica 3.a*. Available at: https://raw.githubusercontent.com/salvacorts/UGR-Metaheuristics/P3/doc/memoria/memoria.pdf (accessed July 03, 2022).

32. *Intel® MPI Library Reference Manual*. Available at: https://www.lrz.de/services/compute/linux-cluster/documentation/protected_nicht_fiona/mpi_30/Reference_Manual.pdf (accessed July 03, 2022).

33. Savin G.I., Shabanov B.M., Telegin P.N., Baranov A.V. Joint supercomputer center of the Russian academy of sciences: Present and future. *Lobachevskii J. of Mathematics*, 2019, vol. 40, pp. 1853–1862. DOI: 10.1134/S1995080219110271.

34. Drezner Z., Hahn P., Taillard E. Recent Advances for the quadratic assignment problem with special emphasis on instances that are difficult for meta-heuristic methods. *Annals of Operations Research*, 2005, vol. 139, pp. 65–94. DOI: 10.1007/s10479-005-3444-z.

35. *Quadratic Assignment Instances*. Available at: http://mistic.heig-vd.ch/taillard/problemes.dir/qap.dir/qap.html (accessed July 03, 2022).

36. Mihic K., Ryan K., Wood A. Randomized decomposition solver with the quadratic assignment problem as a case study. *INFORMS J. on Computing*, 2018, vol. 30, no. 2, pp. 295–308. DOI: 10.1287/ijoc.2017.0781.

Для цитирования

Баранов А.В., Киселев Е.А., Телегин П.Н., Сорокин А.А. Программное средство GraphHunter поиска отображения параллельной программы на структуру суперкомпьютерной системы // Программные продукты и системы. 2022. Т. 35. № 4. С. 583–597. DOI: 10.15827/0236-235X.140.583-597.

For citation

Baranov A.V., Kiselev E.A., Telegin P.N., Sorokin A.A. A GraphHunter software tool for mapping parallel programs to a supercomputer system structure. *Software & Systems*, 2022, vol. 35, no. 4, pp. 583–597 (in Russ.). DOI: 10.15827/0236-235X.140.583-597.