УДК 004.272.26 + 519.157.2 doi: 10.15827/0236-235X.142.607-614 2023. Т. 36. № 4. С. 607–614

Использование библиотеки MPI для параллельной реализации алгоритма полного перебора вариантов

Ай Мин Тайк
С.А. Лупин
Д.А. Федяшин

Ссылка для цитирования
Ай Мин Тайк, Лупин С.А., Федяшин Д.А. Использование библиотеки MPI для параллельной реализации алгоритма полного перебора вариантов // Программные продукты и системы. 2023. Т. 36. № 4. С. 607–614. doi: 10.15827/0236-235X.142.607-614
Информация о статье
Поступила в редакцию: 02.10.2023 После доработки: 16.10.2023 Принята к публикации: 18.10.2023

Аннотация. В работе представлены результаты исследования эффективности параллельного приложения для решения задачи квадратичного назначения. Приложение использует алгоритм полного перебора и интерфейс передачи сообщений (MPI) для организации взаимодействия процессов в распределенной вычислительной среде. Для генерации вариантов решений использован метод лексикографической перестановки, который хорошо распараллеливается и обеспечивает балансировку нагрузки узлов. Для расширения диапазона использования алгоритма перебора на оптимизационные задачи большой размерности пространство поиска решений разбивается на части, что позволяет существенно сократить число рассматриваемых вариантов. Эксперименты проводились в распределенной среде, содержащей 50 узлов с процессорами Intel ® Core™ i5 7-го поколения. Параллельное приложение при решении задачи большой размерности (n = 24) продемонстрировало достижимость ускорения вычислений в диапазоне от 99 до 95 % от максимально возможного, причем во всех случаях было найдено точное решение. Это подтверждает корректность методов, использованных для распределения нагрузки и снижения вычислительной сложности задач. Дальнейшая работа будет направлена на исследование возможности применения предлагаемых подходов при реализации алгоритма перебора и в виде гибридных приложений, что актуально для гетерогенных вычислительных сред. Поскольку задача квадратичного назначения относится к задачам дискретной оптимизации, полученные результаты дают основание считать, что предлагаемые решения будут эффективными и для других задач этого класса.

Ключевые слова: дискретная оптимизация, задача квадратичного назначения, алгоритм полного перебора, распределенные вычисления, MPI, HPC
Благодарности. Исследование проводилось в МИЭТ и МСЦ РАН в рамках государственного задания по теме FNEF-2022-0016

Введение. Люди постоянно сталкиваются с задачами комбинаторной оптимизации, даже не осознавая этого. Большинство систем управления основано на том, что оператор или автомат формирует оптимальное воздействие, выбирая его из некоторого конечного набора возможных решений. Задачи дискретной оптимизации находят широкое применение и в системах автоматизированного проектирования.

В данной статье рассмотрены особенности решения в параллельной среде одной из таких задач – задачи квадратичного назначения (QAP, Quadratic Assignment Problem), которая является характерным представителем этого класса. Она относится к NP-сложным, требующим значительных вычислительных ресурсов. Примерами проблем, формализуемых в виде задачи квадратичного назначения, являются задачи расстановки машин в обрабатывающих центрах, размещения библиотечных элементов в матричных БИС и микросхем на коммутационных платах [1].

Универсальным подходом к решению задач дискретной оптимизации, в частности, и QAP, является алгоритм полного перебора вариантов (BFA, Brute Force Algorithm). Алгоритм BFA методично исследует пространство поиска решений, оценивая все возможные варианты для нахождения точного решения задачи [2]. При этом для перебора вариантов часто используют методы перестановок.

Из-за высокой вычислительной сложности задачи комбинаторной оптимизации, основанные на перестановках, реализуются в распределенных вычислительных средах [3]. Перестановки позволяют проводить поиск наилучшего расположения элементов в задачах, где важны порядок и неповторяемость элементов вектора решения. Методы перестановок реализуются и в распределенных вычислениях. Для взаимодействия узлов вычислителя применяют MPI, стандартный коммуникационный протокол, обеспечивающий обмен информацией между процессорами в распределенной среде [4]. Параллельная реализация BFA позволяет увеличить размер решаемых задач и сократить время вычислений.

Алгоритмические подходы к решению QAP

Рассмотрим несколько известных алгоритмических подходов к параллельной реализации решателей для QAP.

Последовательные алгоритмы эффективны лишь для решения задач небольшой размерности. Для более крупных точные решения могут быть найдены только с использованием методов параллельных вычислений (OpenMP, MPI, Hybrid и CUDA).

В работе [5] представлена комбинаторная оптимизация NP-трудной задачи, приведенной к QAP, с помощью двух параллельных моделей (embarrassingly parallel algorithm и island parallel genetic algorithm) на GPU.

В статье [6] продемонстрировано решение задачи квадратичного назначения с использованием параллельного гибридного алгоритма. Результаты в среднем находятся в пределах 0,05 % от наилучших решений, приведенных в QAPLIB.

В работе [7] при решении QAP использованы графические ускорители. Качество решений, полученных с помощью реализации алгоритмов 2-opt и tabu search, находится в пределах 1,03 % и 0,15 % от известных значений.

Эксперименты дают представление о рабочих характеристиках ускоренных QAP-решателей. В исследовании [8] представлен алгоритм Cooperative Parallel Tabu Search (CPTS) для решения QAP. В CPTS процессоры обмениваются информацией на протяжении всего выполнения алгоритма в отличие от независимых стратегий параллельного поиска, которые агрегируют данные только в конце выполнения. Результаты исследований демонстрируют преимущества параллельных вычислений с точки зрения качества решения, вычислительных затрат и алгоритмической гибкости.

В работе [9] представлен параллельный алгоритм для решения QAP путем включения элементов имитационного отжига в подход локального поиска, способствующий достижению оптимальных результатов. Они сравниваются с результатами множества других алгоритмов, использующих различные экземпляры из набора данных QAPLIB.

Исследование [10] демонстрирует эффективность использования процедуры жадного случайного адаптивного поиска (GRASP) при решении QAP, что в нескольких случаях приводит к почти оптимальным результатам или к достижению оптимальности. Авторы разработали последовательную программу на языке Си, а впоследствии параллельную реализацию для эффективного снижения затрат процессорного времени.

Разработаны и решения задачи QAP с использованием перестановок для поиска экстремума целевой функции. В статье [11] представлен параллельный алгоритм для решения QAP с помощью многопоточного приложения. Для задач размерностью 12 и 14 с помощью перестановок проведен исчерпывающий поиск в пространстве решений. Предложенным методом эти перестановки равномерно распределяются между процессорными ядрами, что позволяет сократить время параллельного выполнения за счет использования шести потоков.

В данной работе исследования направлены на оценку возможности использования результатов, полученных для многопоточных приложений при решении задач комбинаторной оптимизации, на случай распределенных систем, применяющих MPI.

Простейший способ распараллеливания решения проблемы заключается в распределении набора перестановок по различным вычислительным узлам в распределенной вычислительной среде. При этом узлы независимо выполняют вычисления целевой функции для локальных перестановок. Такая стратегия распараллеливания побуждает каждый узел искать свое оптимальное локальное решение. По завершении вычислений на всех узлах выбирается наилучшее решение из всех локальных, которое и будет глобальным экстремумом.

Особенности реализации параллельных вычислений для QAP

Для решения задач QAP большого размера, требующих значительных вычислительных ресурсов, часто применяют методы параллельных вычислений. Такие вычисления могут помочь ускорить процесс решения за счет распределения рабочей нагрузки между несколькими процессорами или узлами, тем самым используя возможности современных высокопроизводительных вычислительных систем. Известны несколько подходов к организации параллельных вычислений, которые могут быть применены и для QAP.

Параллельный локальный поиск. Многие эвристические алгоритмы, такие как имитированный отжиг или генетический алгоритм, могут быть распараллелены путем одновременного запуска нескольких экземпляров алго-

ритма с различными исходными решениями или параметрами [12].

Параллельный метод ветвей и границ. Это широко используемый метод для нахождения точных решений [13]. Распараллеливание построения дерева поиска может привести к значительному ускорению. Узлы дерева поиска могут быть распределены между различными процессорами.

Решатели для целочисленного программирования. Многие коммерческие решатели для целочисленного программирования с открытым исходным кодом, такие как CPLEX и Gurobi, поддерживают параллельное выполнение [14]. Они могут быть использованы и для решения QAP в виде задач смешанного целочисленного программирования.

Алгоритм искусственной пчелиной колонии. Он может решать оптимизационные задачи, такие как QAP. В статье [15] представлена параллельная версия алгоритма для архитектур с разделяемой памятью.

Алгоритм перебора. Ориентированный на системы с общей памятью, он может быть распараллелен с помощью OpenMP, что особенно эффективно для многоядерных процессоров. В работе [16] проведено сравнение последовательной и параллельной реализаций данного алгоритма с использованием OpenMP для решения QAP. Показано, что производительность многопоточного приложения при запуске на четырехъядерном процессоре в 2,3 раза выше, чем последовательного.

Задачу QAP можно разделить на мелкие подзадачи, каждая из них может быть решена независимо на разных процессорах с использованием библиотек передачи сообщений, таких как MPI.

Параллельные гибридные алгоритмы. Они объединяют различные методы решения (эвристические и точные) в параллельной среде, чтобы использовать сильные стороны обоих методов.

Мощные параллельные графические процессоры (GPU) позволяют ускорить определенные вычисления, особенно те, которые связаны с матричными операциями, распространенными в алгоритмах QAP [17].

Важно отметить, что эффективность параллельных вычислений для QAP зависит от различных факторов, включая размер задачи, используемый алгоритм, доступное аппаратное обеспечение и эффективность методов распараллеливания. Параллельные вычисления могут обеспечить значительное ускорение и в то же время привести к возникновению проблем, связанных с балансировкой нагрузки, накладными расходами на связь и синхронизацией. При реализации параллельных алгоритмов для QAP следует учитывать множество факторов: возможность распараллеливания алгоритма, архитектуру доступных вычислительных ресурсов, балансировку нагрузки узлов для достижения наилучшей производительности. Далее представлены результаты исследования эффективности многопоточного приложения, реализующего BFA и использующего библиотеку с MPI, для решения QAP в распределенной вычислительной среде.

Особенности параллельной реализации BFA

Алгоритм полного перебора [18] осуществляет оценку всех возможных перестановок объектов, распределяя их в случае параллельной реализации между процессами с помощью библиотеки MPI.

Параллельная реализация BFA для QAP включает следующие процедуры.

Настройка параметров: задается размерность матрицы связности элементов R и матрицы расстояний D.

Настройка среды и инициализация MPI.

Генерация перестановок: распределяются перестановки между доступными MPI процессами и реализуется метод *следующей лексикографической перестановки* для генерации вариантов решения P.

Вычисление целевой функции: каждый процесс вычисляет целевую функцию для всех перестановок в назначенном ему подмножестве, фиксируя текущий рекорд [16].

Поиск и сравнение конечного результата: ведущий процесс получает со всех узлов локальные минимумы и выбирает из них глобально оптимальное решение.

Завершение MPI и вывод: процесс с rank = 0 выводит решение и затраченное на вычисления время.

В экспериментах использованы два варианта QAP-задачи – размерностью $n = 14$ и $n = 24$. Соответствующие матрицы R и D представлены на сайте (http://www.swsys.ru/uploaded/image/2023-4/5.jpg).

Пространство решения первой задачи было разделено на две части: $n_1 = 9$, $n_2 = 5$, а второй – на четыре: $n_1 = n_2 = n_3 = n_4 = 6$, руководствуясь описанным в [16] методом снижения вычислительной сложности QAP. Число анализируе-

мых вариантов в первом случае составит $P \approx 4{,}4 \cdot 10^7$, а во втором $P \approx 1{,}4 \cdot 10^{17}$.

В описываемом эксперименте использована 64-разрядная версия MPICH2 для реализации MPI. Этот стандарт с открытым исходным кодом (https://www.mpich.org) широко используется для программирования параллельных и распределенных вычислительных систем.

Приведенный далее фрагмент кода иллюстрирует подход к распараллеливанию BFA, применяемый при решении задачи QAP. Приложение реализует генерацию вариантов решения с помощью метода следующей лексикографической перестановки. Программа написана на языке C++ в среде Visual Studio:

```
   1: void reverseOrder(int*p,
int start,int end){
   2: while(start<end)
{swap(p[start],p[end]); start++;end--
;}}
   //генерация функции перестановки
   1:bool permutationfun(int*p, int n)
{int k=-1;
   2:for (i=n-2;i≥0;i--)
{if(p[i]<p[i+1]) {k=i; break;}}
   3: if (k==-1) return false;
int m=n-1;while (p[k]≥p[m]) m--;
   4: swap(p[k],p[m]);
reverseOrder(p,k+1,n-1); return true;}
   // вычислительная функция для каждой
перестановки
   1: int calfun(int*p,int**D,int**R,
int n,int m,int localmin) {calvalue=0;
   2: for i=0 to n do {for j=i+1 to n
do {calvalue+=R[i][j]*D[p[i]][p[j]];}}
   3: if(localmin≥calvalue)localmin=
=calvalue; calvalue=0;
   4: bool hasnext=true; while(hasnext)
{int k=-1;
   5: for (i=m-2; i≥0;i--)
{if(p[i]<p[i+1]) {k=i;break;}}
   6: if (k==-1) hasnext=false;
else {int m=n-1;while(p[k]≥p[m])m--;
   7: swap(p[k],p[m]);
reverseOrder(p,k+1,m-1);
   8: for k=0 to n do {for h=k+1 to n do
{calvalue+=R[k][h]*D[p[k]][p[h]];}}
   9: if(localmin≥calvalue)
localmin=calvalue; return localmin;}
   // основная функция
   1: int main(int argc,char* argv[])
{MPI_Init(&argc,&argv);
   2: MPI_Comm_rank(MPI_COMM_WORLD,
&rank);
   3: MPI_Comm_size(MPI_COMM_WORLD,
&size);
   4: int rank,size,
localmin=5000,calvalue=0;count=0;
double startt,endt;
   5: int ptwo[]={},pthree[]={},
pfour[]={};
   6: int p[]={};
   7: int D[n][n],R[n][n];assign cost
matrix D and distance matrix R;
   8: startt=MPI_Wtime();do{if
(count%size==rank){sort(pthree,
pthree+m);
   9: do{sort(pfour,pfour+m);
do{sort(p,p+m);for(int k=0;k<m;k++){
   10: p[m+k]=ptwo[k];p[(m*2)+k]=
=pthree[k];p[(m*3)+k]=pfour[k];}
   11: localmin=calfun(p,D,R,
localmin);}while(permutationfun
(pfour,m));
   12: }while(permutationfun
(pthree,m));}count++;
while(permutationfun(ptwo,m));
   13: endt= MPI_Wtime();
intglobalmin=0;
   14: MPI_Reduce(&localmin,&globalmin,
1,MPI_INT,MPI_MIN,0,MPI_COMM_WORLD);
   15: if(rank==0)
{cout≪globalmin≪"\t"≪endt-startt;}
MPI_Finalize();return 0;}
```

При решении QAP с использованием параллельного алгоритма BFA библиотека MPI обеспечивает эффективное распределение перестановок и определение глобального оптимума полученных процессами локальных решений.

При параллельном решении QAP нагрузка каждого процесса определяется через его ранг, что позволяет эффективно распределять перестановки и балансировать нагрузку узлов распределенной среды. Функция MPI_Reduce используется для сбора результатов и нахождения глобального оптимума, а MPI_Wtime – для измерения времени выполнения различных частей алгоритма.

Представленный фрагмент программного кода реализует принцип разделения пространства поиска на ограниченные части, описанный в [16]. Для $n = 24$ таких частей четыре, что позволяет разделить нагрузку между процессами, опираясь только на число перестановок в первой части. В рассматриваемом случае оно составляет 720 и определяет максимальное число параллельных процессов. Очевидно, что максимальную эффективность использования вычислительных ресурсов среды можно получить, если это число будет нацело делиться на число запущенных процессов. Для задачи размерностью $n = 14$ пространство поиска можно разделить двумя способами: используя или $n_1 = 9$, $n_2 = 5$, или $n_1 = 5$, $n_2 = 9$.

Генерация вариантов решения с помощью метода следующей лексикографической перестановки гарантирует их неповторяемость для всех процессов.

Такой подход позволяет не только с минимальными накладными расходами организовать параллельные вычисления, но и повысить эффективность распределенной реализации BFA.

Необходимо отметить, что используемый метод распределения нагрузки гарантирует ее равномерность только в том случае, если сложность вычисления критериальной функции не зависит от параметров вектора решения. Это условие выполняется, если n-мерное пространство поиска изотропно по отношению к целевой функции. Для решаемой задачи квадратичного назначения это условие выполняется, если не используются специальные методы. В проводимых вычислениях вопросы оптимизации процесса вычисления целевой функции не рассматривались.

В ходе проведенных экспериментов корректность выбранного метода распределения нагрузки нашла практическое подтверждение. Время работы всех узлов независимо от исследуемого участка пространства поиска решений не отличалось. Аналогичные результаты были получены и для многопоточной реализации BFA [16].

Результаты экспериментов

Для оценки эффективности предлагаемых подходов к параллельной реализации BFA были использованы две QAP-задачи размерностью $n = 14$ и $n = 24$. Условия задач представлены в таблицах (http://www.swsys.ru/uploaded/image/2023-4/5.jpg).

Минимальное значение целевой функции F равно 46 для $n = 14$ и 301 для $n = 24$. Это точные решения, которые были получены для неограниченного пространства поиска.

В таблицах 1 и 2 представлены варианты найденных решений задачи. Для всех решений значение функционала совпадает с известным точным решением, что подтверждает корректность метода разделения пространства поиска. Курсивом выделены номера занимаемых элементами мест.

В таблицах 3 и 4 представлены результаты исследования масштабируемости и эффективности параллельного приложения. Вычислительные эксперименты проводились в компьютерном классе, оснащенном персональными компьютерами с процессором Intel® Core™ i5 7-го поколения, у которых количество физических ядер равно 4, а тактовая частота 3,00 ГГц. В таблицах использованы обозначения: t_{calc} – время вычисления, сек.; N_{proc} – число работающих процессов (использовались только физические ядра); *Acc* – полученное ускорение; *Eff* – эффективность параллельной реализации или коэффициент использования вычислительных ресурсов: $Eff = (t_{calc}(1)/t_{calc}(N_{proc}))\cdot(1/N_{proc}) = Acc\cdot(1/N_{proc})$.

Таблица 1

Решения для $n = 14$

Table 1

Solutions for $n = 14$

Расположение элементов в векторе решения *P*														***F***
1	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***	***12***	***13***	***14***	
1	2	3	4	5	6	7	8	9	10	11	12	13	14	**46**
1	4	7	2	5	8	3	6	9	10	11	12	13	14	
1	2	3	4	5	6	7	8	9	10	11	12	14	13	
1	4	7	2	5	8	3	6	9	10	11	12	14	13	
1	2	3	4	5	6	7	8	9	12	11	10	13	14	
1	4	7	2	5	8	3	6	9	12	11	10	13	14	
1	2	3	4	5	6	7	8	9	12	11	10	14	13	
1	4	7	2	5	8	3	6	9	12	11	10	14	13	

Таблица 2

Решение для $n = 24$

Table 2

Solutions for $n = 24$

Расположение элементов в векторе решения *P*												***F***
1	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***	***12***	
5	2	1	4	3	6	9	8	11	10	7	12	**301**
13	***14***	***15***	***16***	***17***	***18***	***19***	***20***	***21***	***22***	***23***	***24***	
17	14	15	16	13	18	19	24	20	22	23	21	

Таблица 3

Время решения задачи для $n = 14$

Table 3

Problem solution time for $n = 14$

N_{proc}	1	2	3	4	5	6	7	8
t_{calc}	11,48	6,01	4,11	3,08	2,68	2,25	2,36	1,95
Acc	-	1,91	2,79	3,72	4,28	5,10	4,86	5,89
Eff	-	0,96	0,93	0,93	0,86	0,85	0,7	0,74

Таблица 4

Время решения задачи для $n = 24$

Table 4

Problem solution time for $n = 24$

N_{proc}	1	45	90	180
t_{calc}	240213,62	5402,33	2713,76	1407,66
Acc	-	44,46	88,52	170,64
Eff	-	0,99	0,98	0,95

Для чистоты экспериментов при решении задачи с $n = 14$ использовано только по одному ядру на каждом узле.

Следует отметить, что для задачи с $n = 24$ полученные ускорение и коэффициент использования вычислительных ресурсов выше, чем в случае с $n = 14$. Это хорошо согласуется с результатами, полученными и в других работах [19], поскольку доля накладных расходов на синхронизацию процессов будет ниже при большем времени работы узлов. Количество процессов, задействованных в вычислениях для $n = 24$, выбиралось так, чтобы число перестановок в распараллеливаемой части (720) нацело делилось на него.

Заключение

Исследование параллельной реализации алгоритма перебора вариантов при решении QAP-задач позволяет сделать следующие выводы.

Библиотека MPI обеспечивает масштабируемость приложения: высокая эффективность сохраняется при росте числа задействованных вычислительных узлов распределенной среды.

Генерация вариантов решения с помощью метода следующей лексикографической перестановки хорошо распараллеливается и обеспечивает балансировку нагрузки узлов.

Разбиение пространства поиска решений на части позволяет расширить диапазон использования BFA и для оптимизационных задач большой размерности.

В дальнейшем планируется исследовать возможности использования предлагаемых подходов к реализации BFA и в виде гибридных приложений, что, на взгляд авторов, актуально для гетерогенных вычислительных сред. Гибридный параллелизм, использующий комбинацию MPI и OpenMP, позволит более эффективно балансировать нагрузку узлов, содержащих разное количество ядер.

Список литературы

1. Duman E., Or I. The quadratic assignment problem in the context of the printed circuit board assembly process. Comput. & Operations Research, 2007, vol. 34, no. 1, pp. 163–179. doi: 10.1016/j.cor.2005.05.004.

2. Тайк А.М., Лупин С.А., Кхаинг М.Т. Методы повышения эффективности алгоритма полного перебора на примере решения задачи о неограниченном ранце // INJOIT. 2023. Т. 11. № 5. С. 41–46.

3. Pérez-Rodríguez R. An estimation of distribution algorithm for combinatorial optimization problems. IJIO, 2022, vol. 3, no. 1, pp. 47–67. doi: 10.12928/ijio.v3i1.5862.

4. Bhagat L., Arora S., Shrawankar U. Practical Guide to Distributed Systems in MPI. Pothi.com Publ., 2017, 97 p.

5. Gomez E.C., Chaves R.P., Hurtado O.G. Combinatorial optimization NP-Hard problem solved by using the quadratic assignment problem (QAP) solution through a parallel genetic algorithm on GPU. Vision Electronica, 2017, vol. 11, no. 2, pp. 146–151.

6. Tosun U. On the performance of parallel hybrid algorithms for the solution of the quadratic assignment problem. Eng. Applications of Artificial Intelligence, 2015, vol. 39, pp. 267–278. doi: 10.1016/j.engappai.2014.12.011.

7. Sonuc E., Sen B., Bayir S. A cooperative GPU-based parallel multistart simulated annealing algorithm for quadratic assignment problem. JESTECH, 2018, vol. 21, no. 5, pp. 843–849. doi: 10.1016/j.jestch.2018.08.002.

8. Abdelkafi O., Derbel B., Liefooghe A. A parallel tabu search for the large-scale quadratic assignment problem. Proc. IEEE CEC, 2019, pp. 3070–3077. doi: 10.1109/CEC.2019.8790152.

9. Ozcetin E., Ozturk G. A parallel iterated local search algorithm on GPUs for quadratic assignment problem. IJET, 2018, vol. 4, no. 2, pp. 123–127.

10. Loranca B.B., Analco M.E., Velázquez R.G., López A.S. et al. Quadratic assignment problem: A solution approach with parallel GRASP. IJCOPI, 2017, vol. 8, no. 3, pp. 33–38.

11. Cardenas E.G., Poveda R.Ch., Garcia O.H. A solution for the quadratic assignment problem (QAP) through a parallel genetic algorithm based grid on GPU. Appl. Math. Sci., 2017, vol. 11, no. 57, pp. 2843–2854. doi: 10.12988/ams.2017.710319.

12. Codognet Ph., Munera D., Diaz D., Abreu S. Parallel Local Search. In: Handbook of Parallel Constraint Reasoning. Hamadi Y., Sais L. (eds), 2018, pp. 381–417. doi: 10.1007/978-3-319-63516-3_10.

13. Fujii K., Ito N., Kim S., Kojima M., Shinano Y., Toh K.-Ch. Solving Challenging Large Scale QAPs, ZIB Report, 2021, art. 21-02, pp. 1–15.

14. Ralphs T., Shinano Y., Berthold T., Koch T. Parallel solvers for mixed integer linear optimization. In: Handbook of Parallel Constraint Reasoning, 2018, pp. 283–336. doi: 10.1007/978-3-319-63516-3_8.

15. Li S., Li W., Zhang H., Wang Z. Research and implementation of parallel artificial bee colony algorithm based on ternary optical computer. Automatika, 2019, vol. 60, no. 4, pp. 422–431. doi: 10.1080/00051144.2019.1639118.

16. Тайк А.М., Лупин С.А., Балабаев С.А. Особенности применения алгоритма полного перебора для решения задачи квадратичного назначения // INJOIT. 2023. Т. 11. № 7. С. 60–68.

17. Kumar M., Mitra P. Solving quadratic assignment problem using iterated local search on GPU spatial memory, Proc. Int. Conf. ACIA, 2023, vol. 2705, no. 1, pp. 1–9. doi: 10.1063/5.0133414.

18. Thike A.M., Lupin S. Parallel application for wireless network topology optimisation. IJEEDC, 2019, vol. 7, no. 4, pp. 14–20.

19. Рыбаков А.А., Мещеряков А.О. Векторизация трехмерного метода погруженных границ для повышения эффективности расчетов на микропроцессорах Intel // Программные продукты и системы. 2023. Т. 36. № 1. С. 130–143. doi: 10.15827/0236-235X.141.130-143.

Software & Systems doi: 10.15827/0236-235X.142.607-614 2023, vol. 36, no. 4, pp. 607–614

Using MPI library for parallel implementation of a brute-force algorithm

Aye Min Thike
Sergey A. Lupin
Dmitry A. Fedyashin

For citation
Thike, A.M., Lupin, S.A., Fedyashin, D.A. (2023) 'Using MPI library for parallel implementation of a brute-force algorithm', *Software & Systems*, 36(4), pp. 607–614 (in Russ.). doi: 10.15827/0236-235X.142.607-614
Article info
Received: 02.10.2023 After revision: 16.10.2023 Accepted: 18.10.2023

Abstract. The paper presents results of studying the effectiveness of a parallel application for solving a quadratic assignment problem. The application uses a brute-force algorithm and a Message Passing Interface (MPI) to ensure interprocess communication in a distributed computing environment. Solution options are generated by a lexicographic permutation method, which is well parallelized and ensures load balancing of nodes. The range of brute-force algorithm use to large scale optimization problems is expanded by dividing a search space into parts, which significantly reduces the number of options under consideration. The experiments were conducted in a distributed environment containing 50 nodes with the 7th generation Intel ® Core ™ i5 processors. A parallel application for solving a large-scale problem (n = 24) demonstrated the possibility accelerating calculations in the range from 99 to 95 % from the maximum possible one; an exact solution was found in all cases. This confirms the correctness of methods used for load distribution and for computational complexity reduction of tasks. Further work will be aimed at exploring the possibility of using the proposed approaches to implementing a brute-force algorithm and as hybrid applications, which is relevant for heterogeneous computing environments. Since a quadratic assignment problem is a discrete optimization problem, the obtained results suggest that the proposed approaches are effective for other problems of this class.

Keywords: discrete optimization, quadratic assignment problem, brute-force algorithm, distributed computing, MPI, HPC
Acknowledgements. The study was conducted at MIET and JSCC RAS within the framework of the state assignment on the topic FNEF-2022-0016

References

1. Duman, E., Or, I. (2007) 'The quadratic assignment problem in the context of the printed circuit board assembly process', *Comput. & Operations Research*, 34(1), pp. 163–179. doi: 10.1016/j.cor.2005.05.004.

2. Thike, A.M., Lupin, S.A., Khaing, M.T. (2023) 'Methods for improving the efficiency of Brute-force algorithm by the example of solving an Unbounded Knapsack Problem', *INJOIT*, 11(5), pp. 41–46 (in Russ.).

3. Pérez-Rodríguez, R. (2022) 'An estimation of distribution algorithm for combinatorial optimization problems', *IJIO*, 3(1), pp. 47–67. doi: 10.12928/ijio.v3i1.5862.

4. Bhagat, L., Arora, S., Shrawankar, U. (2017) *Practical Guide to Distributed Systems in MPI*. Pothi.com Publ., 97 p.

5. Gomez, E.C., Chaves, R.P., Hurtado, O.G. (2017) 'Combinatorial optimization NP-Hard problem solved by using the quadratic assignment problem (QAP) solution through a parallel genetic algorithm on GPU', *Vision Electronica*, 11(2), pp. 146–151.

6. Tosun, U. (2015) 'On the performance of parallel hybrid algorithms for the solution of the quadratic assignment problem', *Eng. Applications of Artificial Intelligence*, 39, pp. 267–278. doi: 10.1016/j.engappai.2014.12.011.

7. Sonuc, E., Sen, B., Bayir, S. (2018) 'A cooperative GPU-based parallel multistart simulated annealing algorithm for quadratic assignment problem', *JESTECH*, 21(5), pp. 843–849. doi: 10.1016/j.jestch.2018.08.002.

8. Abdelkafi, O., Derbel, B., Liefooghe, A. (2019) 'A parallel tabu search for the large-scale quadratic assignment problem', *Proc. IEEE CEC*, pp. 3070–3077. doi: 10.1109/CEC.2019.8790152.

9. Ozcetin, E., Ozturk, G. (2018) 'A parallel iterated local search algorithm on GPUs for quadratic assignment problem', *IJET*, 4(2), pp. 123–127.

10. Loranca, B.B., Analco, M.E., Velázquez, R.G., López, A.S. et al. (2017) 'Quadratic assignment problem: A solution approach with parallel GRASP', *IJCOPI*, 8(3), pp. 33–38.

11. Cardenas, E.G., Poveda, R.Ch., Garcia, O.H. (2017) 'A solution for the quadratic assignment problem (QAP) through a parallel genetic algorithm based grid on GPU', *Appl. Math. Sci*., 11(57), pp. 2843–2854. doi: 10.12988/ams.2017.710319.

12. Codognet, Ph., Munera, D., Diaz, D., Abreu, S. (2018) 'Parallel local search', in Hamadi, Y., Sais, L. (eds) *Handbook of Parallel Constraint Reasoning*, pp. 381–417. doi: 10.1007/978-3-319-63516-3_10.

13. Fujii, K., Ito, N., Kim, S., Kojima, M., Shinano, Y., Toh, K.-Ch. (2021) 'Solving challenging large scale QAPs', *ZIB Report*, art. 21-02, pp. 1–15.

14. Ralphs, T., Shinano, Y., Berthold, T., Koch, T. (2018) 'Parallel solvers for mixed integer linear optimization', in *Handbook of Parallel Constraint Reasoning*, pp. 283–336. doi: 10.1007/978-3-319-63516-3_8.

15. Li, S., Li, W., Zhang, H., Wang, Z. (2019) 'Research and implementation of parallel artificial bee colony algorithm based on ternary optical computer', *Automatika*, 60(4), pp. 422–431. doi: 10.1080/00051144.2019.1639118.

16. Thike, A.M., Lupin, S.A., Balabaev, S.A. (2023) 'The features of using a brute force algorithm for solving a quadratic assignment problem', *INJOIT*, 11(7), pp. 60–68 (in Russ.).

17. Kumar, M., Mitra, P. (2023) 'Solving quadratic assignment problem using iterated local search on GPU spatial memory', *Proc. Int. Conf. ACIA*, 2705(1), pp. 1–9. doi: 10.1063/5.0133414.

18. Thike, A.M., Lupin, S. (2019) 'Parallel application for wireless network topology optimisation', *IJEEDC*, 7(4), pp. 14–20.

19. Rybakov, A.A., Meshcheryakov, A.O. (2023) 'Vectorization of the three-dimensional immersed boundary method for improving the efficiency of calculations on Intel microprocessors', *Software & Systems*, 36(1), pp. 130–143 (in Russ.). doi: 10.15827/0236-235X.141.130-143.

Авторы

Ай Мин Тайк [1], к.т.н., докторант, ayeminthike52@gmail.com
Лупин Сергей Андреевич [1, 2], к.т.н., профессор, lupin@miee.ru
Федяшин Дмитрий Андреевич [1], начальник ВЦ, fedyashinda@gmail.com

[1] Национальный исследовательский университет «МИЭТ», г. Москва, 124498, Россия
[2] Межведомственный суперкомпьютерный центр РАН, г. Москва, 119334, Россия

Authors

Aye Min Thike [1], Cand. of Sci. (Engineering) Doctoral Student, ayeminthike52@gmail.com
Sergey A. Lupin [1, 2], Cand. of Sci. (Engineering), Professor, lupin@miee.ru
Dmitry A. Fedyashin [1], Head of Computer Center, fedyashinda@gmail.com

[1] National Research University of Electronic Technology, MIET, Moscow, 124498, Russian Federation
[2] Joint Supercomputer Center of RAS, Moscow, 119334, Russian Federation