УДК 004.04

РАЗРАБОТКА СИСТЕМЫ ПРАВИЛ ПРЕПРОЦЕССОРА С ЦЕЛЬЮ ОПТИМИЗАЦИИ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ НАУЧНО-ТЕХНИЧЕСКОГО ХАРАКТЕРА ПО ВРЕМЕНИ ВЫПОЛНЕНИЯ ДЛЯ ВЫЧИСЛИТЕЛЕЙ АРХИТЕКТУРЫ MPP

Палагин В.В., соискатель, МГУПИ, Москва, Россия.
E-mail: art.stritn@gmail.com

Аннотация. В работе обсуждаются вопросы эффективного распределения исходных данных по вычислительным узлам (ВУ) многопроцессорной вычислительной системы (МВС) архитектуры MPP с целью минимизации коммуникационного времени. Предлагается формальная модель процесса, реализующего такое распределение, на уровне исходных текстов программ и методика оптимизации.
Ключевые слова. Параллельное программирование, архитектура MPP, распределение данных по ВУ, выбор оптимального (рационального) варианта распределения.

DEVELOP A SYSTEM OF RULES PREPROCESSOR WITH THE GOAL OF OPTIMIZATION OF PARALLEL PROGRAMS FOR SCIENTIFIC AND TECHNICAL EXECUTION TIME FOR CALCULATORS MPP ARCHITECTURE

Palagin V.V., postgraduate student, MGUPI, Moscow, Russia.
E-mail: art.stritn@gmail.com

Abstract. The paper discusses the efficient allocation of initial data array in a distributed memory multiprocessor computer systems (MVS) MPP architecture to minimize downtime calculators during communication between nodes MVS. The formal model of the process of implementing such a distribution, source-level programs and optimization technique.
Keywords. Concurrent programming, architecture MPP, data distribution via local storage resources of compute nodes, the choice of the optimal (rational) distribution.

В силу всё более возрастающей необходимости решения национально-значимых задач в приемлемое время, развитие суперкомпьютерных вычислений является важным направлением современной науки. В то же время приоритет аппаратных платформ для параллельных вычислений сместился в сторону кластерных систем, реализующих архитектуру Massive Parallel processing (MPP) [1] и составляющих по данным проекта Top 500 [2] более 80% от общего числа суперкомпьютеров мира. В связи с этим при разработке программного обеспечения (ПО) для многопроцессорных вычислительных систем (МВС) с распределённой памятью, прикладные специалисты часто применяют метод достижения параллелизма, при котором единая программа обрабатывает распределённые данные - single program, multiple data (SPMD) [3]. Во время выполнения такой программы её процессы разделены и запускаются параллельно на

нескольких вычислительных узлах (ВУ) МВС, работая каждый в своей локальной памяти.

Для обеспечения обмена данными между процессами параллельной программы, используются низкоуровневые механизмы передачи сообщений – Message Passing Interface (MPI) [4]. В этом случае на каждом ВУ выполняется рабочий процесс (slave process), который является частью основной программы, запущенной на управляющем узле (master process), и взаимодействует с другими узлами через получение/отправку (MPI_*recv/MPI_*send) данных (рис. 1).

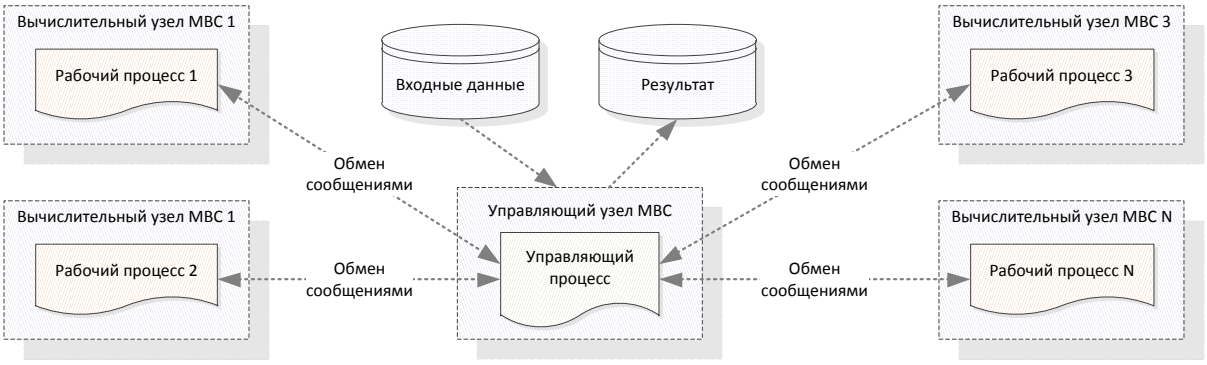

Рис. 1. Общая схема организации вычислений для МВС с распределённой памятью

Для параллельных программ, реализующих такую организацию параллельных вычислений, общее время выполнения складывается из времени обмена сообщениями по коммуникационной сети и время выполнения процессов локально на ВУ, отвечающих за основной алгоритм вычислений:

$$T_{global} = T_{net} + T_{local}, \quad (1)$$

где T_{global} – общее время выполнения параллельной программы;

T_{net} – время сетевого обмена данными между ВУ;

T_{local} – время выполнения локальных вычислений на ВУ.

Время сетевого обмена, без учёта зависимости от характеристик аппаратной платформы МВС, в общем случае определяется числом пересылок и объёмом пересылаемых сообщений между ВУ в процессе вычислений, которые заданы целевым алгоритмом, а также начальным распределением (инициализацией) исходных данных. Время локального выполнения подпрограммы на ВУ зависит от представления массивов в оперативной памяти (ОП) для разных языков программирования (Fortran/C/C++), а также таких специфических характеристик как алгоритм работы кэш памяти и т.д. Для параллельных программ научно-технического характера,

реализующих сеточные методы в задачах моделирования различных процессов в физике/механике, основная трудоёмкость и время выполнения сосредоточено во вложенных циклах (cycle nest) [5], обрабатывающих массивы большой размерности (более 10^5 элементов).

Эффективность оптимизации распределения данных в параллельной программе с целью уменьшения глобального времени выполнения будет определяться, в том числе классом решаемых задач. Для параллельных программ, у которых $T_{net} \gg T_{local}$, подбор оптимального размещения данных в распределённой памяти МВС даст ощутимый эффект. Характерный пример – подкласс научно-технических задач, реализующий сеточные задачи с операциями над массивами большой размерности при незначительной трудоёмкости расчётных формул в гнезде циклов (ГЦ).

С ростом размерности задачи и числа ВУ, чем характеризуются современные суперкомпьютерные вычисления, на первый план выходит так называемый эффект «стены памяти», при котором разрыв по времени между вычислениями на современных процессорах и доступом к данным, расположенным в локальной памяти узлов, через коммуникационную среду становится определяющим препятствием в достижении экзафлопсного барьера. По этой причине актуальность вектора исследования направленного в сторону минимизации эффекта «стена памяти» для параллельных программ средствами оптимизации распределения данных является обоснованной.

В данной работе, с учётом специфики рассматриваемого класса задач, структура параллельной программы представлена как совокупность блоков исходного кода с набором ГЦ. В рамках блока средствами программных механизмов, которые поддерживаются большинством современных систем параллельного программирования (DVM/mpC/HOPMA), размещение данных задаётся единожды на всё время выполнения. Распределение данных обозначим как список [5]:

$$D_K = \{d_1, d_2,..,d_k,..,d_K\}, \quad (2)$$

где d_k – элемент списка D_K, описывающий заданное распределение;

K – общее число вариантов распределения.

Блок обозначим, как совокупность ГЦ, и представим в виде списка:

$$F_M = \{f_1, f_2,..,f_m,..,f_M\}, \quad (3)$$

где f_m – элемент списка F_M, описывающий ГЦ m;

M – общее число сгруппированных в блоке ГЦ.

Согласно принятым обозначениям поиск оптимального размещения данных для распределённой памяти МВС в рамках блока производится на основе специализированной системы правил (рис. 2).

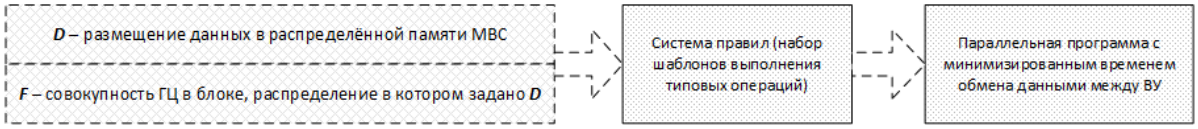

Рис. 2. Общая схема поиска оптимального распределения данных в целевой параллельной программе на основе системы правил.

Принципиально возможны три подхода к варьированию входных параметров *D* и *F*, где символами *var* и *const* обозначены действия изменения и постоянства соответственно:

1. *D =var, F = const* – поиск распределения данных с целью минимизации времени выполнения блока – подход представленный в данной работе;
2. *D = const, F = var* – эквивалентные преобразование ГЦ в блоке при неизменном параметре распределения – подход характерный для таких распараллеливающих/оптимизирующих систем как FORGExplorer/V-Ray/OPC [6];
3. *D = var, F = var* – изменению подвергаются как распределение, так и ГЦ в рамках целевого блока, что является наиболее сложным вариантом оптимизационной задачи.

Тогда общий вид функции при решение задачи уменьшения общего времени выполнения параллельной программы с учётом варьируемых параметров и характеристик аппаратной платформы будет следующим:

$$Z = f(D, F, Q_{node}, X_{node}, Y_{net}), \quad (4)$$

где Q_{node} – число задействованных в процессе обработки данных ВУ;

X_{node} – характеристики аппаратного обеспечения ВУ;

Y_{net} – характеристики коммуникационной сети.

Учитывая специфику целевого класса задач определим время выполнения параллельной программы T_{prog} с учётом распределения времени выполнения по блокам T_{block}:

$$T_{prog} = \sum_{b=1}^{B}\left(T_{block,b}\right) + T_{block}^{ext}, \quad (5)$$

где $b \in \{1,..,B\}$ – число блоков с набором ГЦ в программе,

T_{block}^{ext} – время выполнения программного кода за пределами блоков. Соответственно, время выполнения блока T_{block} формируется из времени выполнения ГЦ с зависимостью по данным T_{nest} и программного кода за пределами циклов T_{nest}^{ext}:

$$T_{block} = \sum_{m=1}^{M}\left(T_{nest,m}\right) + T_{nest}^{ext}, \quad (6)$$

где $m \in \{1,..,M\}$ – число ГЦ с зависимостью по данным в блоке.

Исходя из специфики целевого класса программ, для которых основная трудоёмкость сосредоточена в циклах, величины T_{block}^{ext} и T_{nest}^{ext} примем как незначимые с точки зрения времени выполнения. По причине того, что в каждом теле циклов выполняются как операции индексации при обращении к элементам массивов, так и чисто вычислительные операции, необходимо применять комплексный подход к анализу трудоёмкостей выполнения каждого ГЦ. Для информации о трудоёмкости ГЦ дополним каждый элемент F компонентами, через которые получено время однократного выполнения $t_{m,l}$ тела цикла m глубиной $l \in \{1..l_M\}$:

$$t_{m,l} = (N_{calc} \cdot T_{calc}) + (N_{mem} \cdot T_{mem}), \quad (7)$$

где N_{calc} – число арифметических операций,

T_{calc} – время выполнения арифметической операции,

N_{mem} – число операций индексирования (доступа к данным),

T_{mem} – время выполнения операции индексирования.

Величины N_{calc} и N_{mem} определяются реализацией целевого алгоритма, в то время как T_{mem} определяется расположением данных в локальной памяти ВУ или в памяти другого узла. Полное время выполнения тела ГЦ m глубиной l:

$$T_{m,l} = t_{m,l} \cdot ((i_2 - i_1)) / i_3, \quad (8)$$

где i_1, i_2, i_3 – начальное, конечное значение и приращение индексной переменной.

В случае неполного выполнения ГЦ при использовании операторов условного перехода формализация значительно усложняется. Время выполнения ГЦ $T_{nest,m}$ без учёта досрочного выхода из цикла будет определяться:

$$T_{nest,m} = \sum_{s=1}^{l_m} \left[\prod_{n=1}^{l_m} \left(T_n \cdot {(i_{2,n} - i_{1,n})}/{i_{3,n}} \right) \right], \quad (9)$$

где T_n – время выполнения тела цикла на глубине $n(s)$ целевого ГЦ.

Для того, что бы обеспечить независимость от аппаратной реализации МВС величины T_{calc} и T_{mem} задаются в относительных единицах времени выполнения. Тогда, определив T_{mem} для каждой операции доступа к данным по индексу при заданном распределении D, получим функцию цели для поставленной задачи $T_{block} = f_B(d_1, d_2, .., d_k) \rightarrow min$. Таким образом, уменьшение времени выполнения целевого блока параллельной программы достигается поиском минимума в конечном списке D, хранящем все варианты распределения. Программной реализацией данного алгоритма является препроцессор для метаязыка [8].

По причине невысокой трудоёмкости лексико-синтаксического анализа по сравнению с классическими языками программирования в качестве инструмента

модификации исходного кода целевого класса программ выбран язык НОРМА, разработанный в ИПМ РАН им. М.В. Келдыша [9]. В ходе препроцессирования исходного кода целевой программы из входной последовательности выделяются необходимые лексемы (токены) и строится абстрактное синтаксическое дерево (АСД), в котором внутренние вершины сопоставлены с блоками, а листья – с ГЦ. Таким образом, целевая параллельная программа представляется в виде иерархии блоков исходного кода с набором ГЦ, где вершиной является точка входа в программу. На основе сгенерированной системы правил модификации подвергаются операторы метаязыка, отвечающие за распределение данных по ВУ.

Система правил, позволяющая сделать вывод об оптимальности целевого варианта распределения по критерию времени выполнения, представляет собой набор шаблонов выполнения типовых операций над матрицами/векторами (матрица – двухмерный массив, вектор - одномерный массив), которые наиболее часто используются в параллельных программах для научно-технических расчётов. Под типовыми операциями подразумевается умножение/сложением матриц/векторов, поиск максимума/минимума в матрице/векторе, умножение/сложение элементов матрицы/вектора, транспонирование матрицы, вычисление обратной матрицы, решение СЛАУ.

В свою очередь шаблон выполнения представляет собой значение оптимального распределения данных в рамках блока для каждой типовой операции, полученное из практических экспериментов на различных аппаратных платформах МВС (рис. 3).

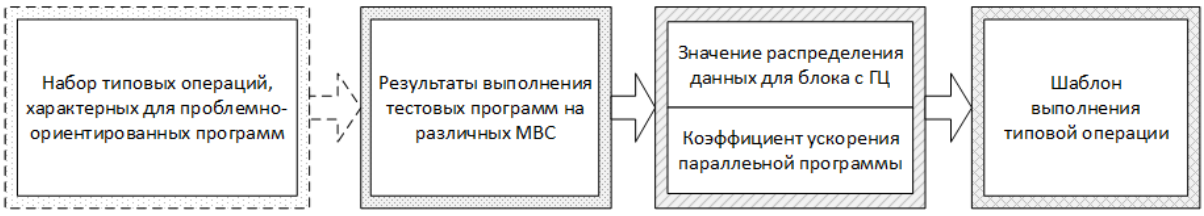

Рис. 3. Формирование системы правил для определения оптимальности распределения данных

Типовая операция характеризуется структурой построения ГЦ – глубиной вложенности, порядком следования индексов, набором арифметических операций в теле цикла. Для каждой тестовой параллельной программы, реализующей одну типовую операцию в рамках блока, опытным путём было получено время выполнения в относительных единицах (для устранения привязки к конфигурации аппаратной платформы), зависящее от выбранного параметра распределения данных. В ходе обработке результатов времени выполнения тестовых параллельных программ,

реализующих набор типовых операций над матрицами, в шаблон добавлялся коэффициент ускорения, выражающийся отношением наилучшего к наихудшему времени выполнения в зависимости от заданного распределения (рис. 4).

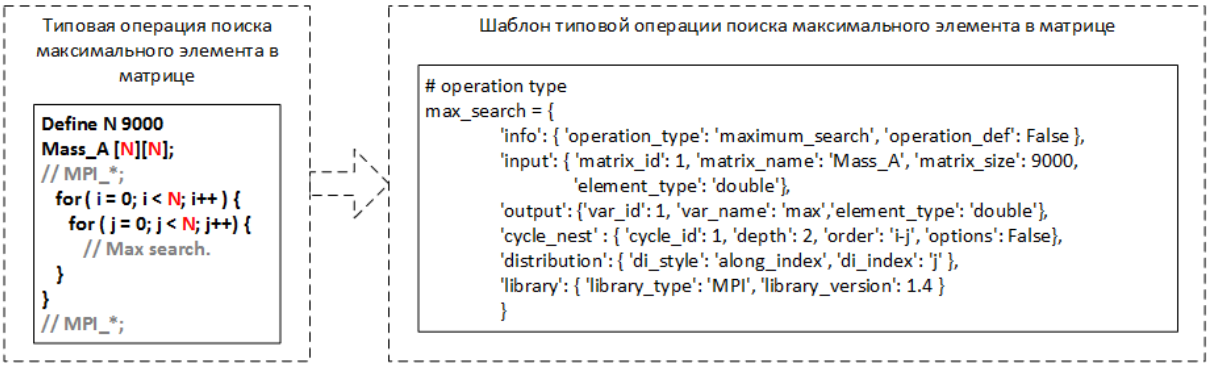

Рис. 4. Пример формирования шаблона типовой операции поиска максимального элемента в матрице

На первом этапе разработки препроцессора и формирования системы правил тестирование проводилось на аппаратной платформе МГУПИ [10]. Структура получаемого исходного кода и распределение времени выполнения в используемых программах соответствуют характеру параллельных программ для решения сеточных задач математической физики/механики, задействованных в промышленных целях.

В качестве первого примера рассмотрим параллельную программу, реализующую типовые операции поиска максимального элемента в квадратной матрице размерностью $3*10^3$ элементов и умножение матрицы на вектор длинной $2*10^3$ элементов (вещественные числа двойной точности в Фортран – DOUBLE PRECISION). При получение исполняемого (бинарного) кода параллельных программ с различными вариантами размещения массивов в распределённой памяти вычислителя, алгоритм описывался средствами метаязыка. Метаязыковой исходный код обрабатывался препроцессором PyNorma [8] в режиме перебора значений (--generate-test). В данном режиме работы препроцессор определяет все возможные варианты распределения *D* и генерирует на каждый вариант отдельную параллельную программу. Затем версии параллельных программ на метаязыке транслировались в исходный код на Фортран с вызовами MPI функций, который компилировался в исполняемый код Executable and Linkable Format (ELF) [11] для выполнения на ОС Slackware Linux [12]. Перед компиляцией в исходный код программ на Фортран внедрялись специальные счётчики, использующие системные вызовы ядра ОС Linux, задачей которых являлось вычисление времени выполнения и запись полученных результатов в отдельный файл. Компиляция осуществлялась командой mpif77 -o <BIN_NAME> <SOURCE_NAME.F>,

а запуск производился командой mpirun -n <PROCESSORS_COUNT+1> <BIN_NAME>, где BIN_NAME – имя файла скомпилированной программы, SOURCE_NAME.F – имя файла с исходным кодом Фортран, PROCESSORS_COUNT – число ВУ для обсчёта вычислительной задачи без учёта управляющего ВУ.

На рис. 5 приведен график (сплошной линией отмечены версии программ, реализующие типовую операцию умножения матрицы на вектор, пунктиром – поиск максимального элемента в матрице), отражающий зависимость коэффициента ускорения тестовой параллельной программы от значения параметра распределения с задействованием от 1 до 20 ВУ.

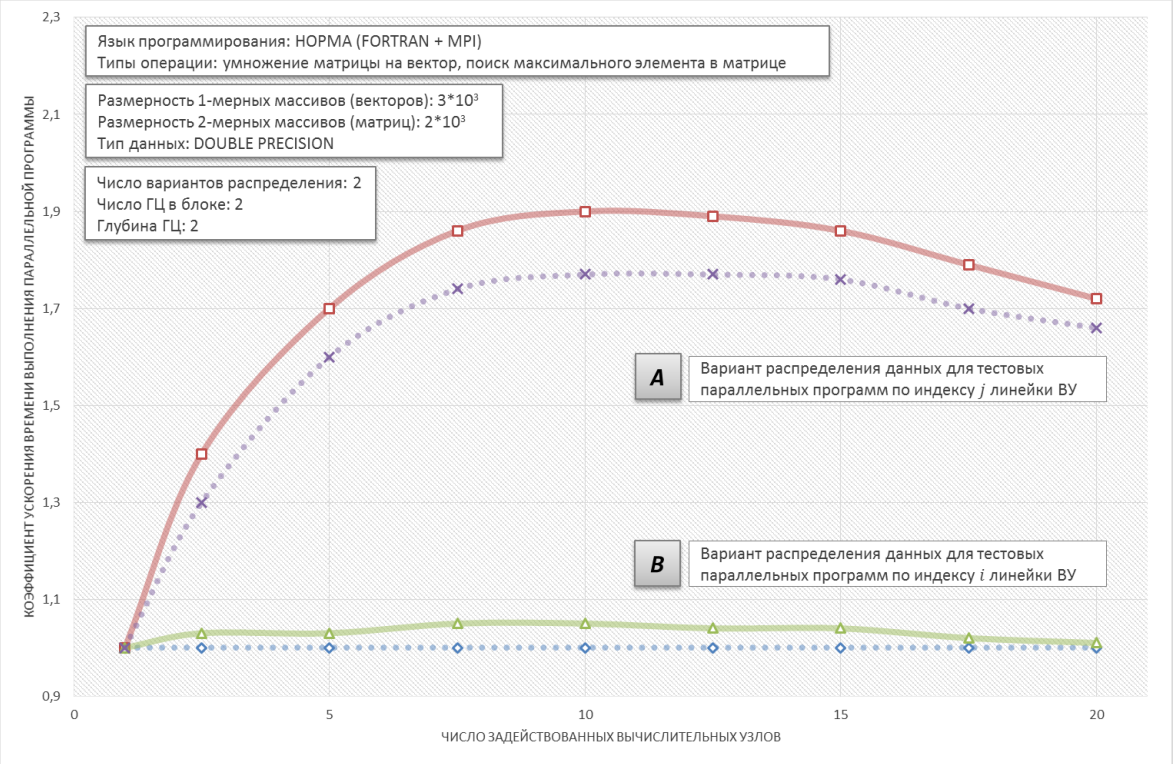

Рис. 5. Зависимость коэффициента ускорения параллельных программ, реализующих различные типовые операции, от варианта распределения данных

Как иллюстрирует график, в целом положительная тенденция сохраняется, и версии параллельной программы с рациональным вариантом распределения исходных данных по ВУ имеют коэффициент ускорения в среднем 2,2 больше. Для данной задачи при распределении исходных массивов в блоке с набором ГЦ (основной и служебный ГЦ глубиной вложения 2) по индексу *j* наблюдается явный прогресс.

Следует отметить, что программы для научно-технических расчётов используемые в промышленности для вычислений данных большого размерности по сравнению с приведённым примером имеют весьма сложную логику и структуру исходного кода. Дополнительную погрешность вносит использованная для проведения

серии тестов специфика аппаратной платформы вычислительного кластера МГУПИ, в силу ограничений на количество ВУ и задержки/скорость передачи данных в системной сети по сравнению с промышленными решениями (InfiniBand, Myrinet). Кроме того, специфика реализации конечно-разностных методов решения краевых задач в некоторых случаях может сделать невозможным перераспределение данных без глобальной модификации исходного кода параллельной программы, что по трудоёмкости для прикладного специалиста эквивалентно полному перепроектированию структуры программы. Поэтому, для того, чтобы в реальных условиях оценить эффективность работы препроцессора был проведён ряд экспериментов по оптимизации параллельных программ, обрабатывающих значительные объёмы данных, на аппаратной платформе МГУ им. Ломоносова [13].

Поэтому в качестве второго примера рассмотрим параллельную программу, реализующую решение системы линейных алгебраических уравнений (СЛАУ) методом Гаусса-Жордана. В качестве входных данных использовались квадратные матрицы размерностью до $4*10^5$ элементов, типом данных элементов массива – вещественные числа двойной точности (DOUBLE PRECISION).

На рис. 6 приведен график (сплошной линией отмечены версии программ относящихся к группе C, пунктиром – B, линией точек – A), отражающий зависимость ускорения целевой параллельной программы от значений параметра распределения с задействованием от 2 до 64 ВУ.

По аналогии с первым примером результаты времени выполнения параллельных программ, приведённые на графике, были объединены в три группы (*A, B, C*). График построен в виде зависимости ускорения от параметра распределения, что даёт возможность абстрагироваться от абсолютных величин и характеристик аппаратной платформы МВС. В группу *A* объединялись результаты для вариантов параллельных программ с вариантами распределения в блоке исходного кода *jik,ji,ji/jki,ij,ij* имеющие близкие значения по времени выполнения (препроцессировнный на основе системы правил исходный код). Для групп *B* и *C* значения индексов были *ikj,ij,ij/ijk,ij,ij* и *kji,ji,ji/kij,ji,ji* соответственно.

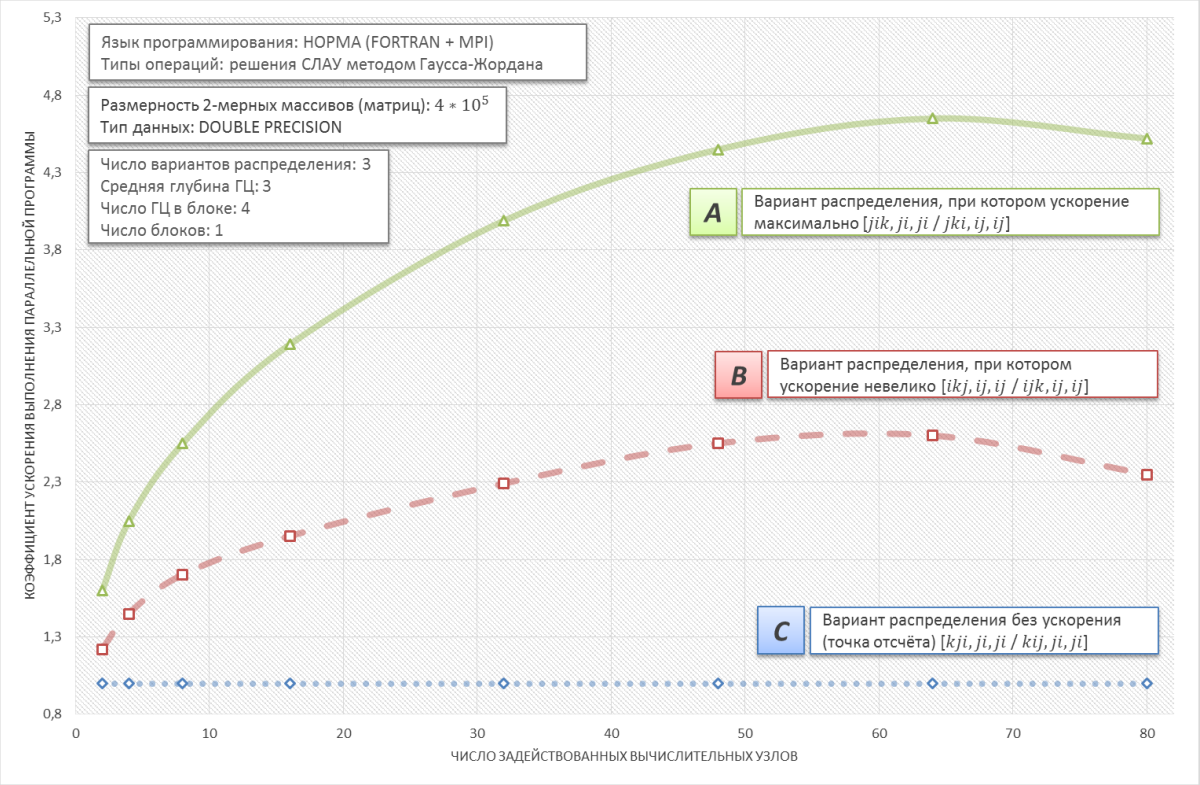

Рис. 6. Зависимость ускорения параллельной программы от варианта распределения массива данных по ВУ на суперкомпьютере «Ломоносов»

При продлении порога в 65 задействованных ВУ, время выполнения для целевых программ стабилизируется и остаётся неизменным с наращиванием количества ВУ до 75. Ускорение программы на этом отрезке между 65 и 75 задействованными ВУ для всех групп сохраняется и составляет 1,85 раза для наилучшего и среднего результата и в 4,45 раза для наилучшего и наихудшего результата.

В качестве заключения необходимо отметить, что создание системы правил, на основе которой принимается решение о рациональном характере заданного распределения данных в программе, требует проведения значительного числа экспериментов на различных аппаратных платформах МВС. Существуют также ограничения на набор типовых операций, которые могут быть оценены с помощь подобной системы правил, что накладывает отпечаток на целевой класс параллельных программ, который может содержать ограниченное количество различных вариаций ГЦ. Несмотря на отмеченные ограничения, применение предложенной стратегии оптимизации в некоторых случаях позволяет добиться прироста производительности в 4-5 раз, в зависимости от числа задействованных ВУ для параллельных программ, обрабатывающих массивы большой размерности.

Список литературы.

1. David B. Kirk, Wen-mei W. Hwu. Programming Massively Parallel Processors. // Morgan Kaufmann, 1 edition. 2012. 280p.
2. Статистика по 500 самых мощных компьютеров мира. [Электронный ресурс] // Top 500: сайт. – URL: http://i.top500.org/stats (дата обращения 22.02.2014).
3. David A. Patterson, John L. Hennessy. Computer Organization and Design, Fourth Edition. // Morgan Kaufmann, 4 edition. 2011. 914p.
4. Rainer Keller, Edgar Gabriel, Jack Dongarra, Michael M. Resc. Recent Advances in the Message Passing Interface. // Springer 2011 edition. 2011. 358p.
5. Christopher Grafton. Mastering Hurst Cycle Analysis: A modern treatment of Hurst's original system of financial market analysis. // Harriman House. 2011. 500p.
6. Палагин В.В. К вопросу об ускорении параллельных программ научно-технических расчётов путём использования архитектурных особенностей многопроцессорных вычислительных систем. // Журнал «Программная инженерия». М.: Издательство «Новые технологии», 2013. №2. С. 21-25.
7. Средства автоматического распараллеливания [Электронный ресурс] // Все о мире суперкомпьютеров и параллельных вычислений: сайт. — URL: https://parallel.ru/tech/tech_dev/auto_par.html (дата обращения 23.02.2014).
8. Палагин В.В. Использование препроцессора Pynorma с целью оптимизации параллельных программ для научно-технических расчётов по времени выполнения. // Сборник научных статей по итогам международной заочной научно-практической конференции «Современные тенденции экономики, управления, права, социологии, образования, медицины, физики, математики». – СПб.: Издательство «КультИнформПресс», 2013. С. 230-233.
9. Спецификация языка НОРМА. [Электронный ресурс] // Язык программирования НОРМА: сайт. – URL: http://www.keldysh.ru/pages/norma/specif/book1.htm (дата обращения 15.11.2013).
10. Баканов В. М. Априорная количественная оценка эффективности параллельных программ на конкретных многопроцессорных системах // Программная инженерия. 2011. № 1. С. 34-38.
11. Yoann Padioleau. Parsing C/C++ code without preprocessing. // Proceeding CC '09 Proceedings of the 18th International Conference on Compiler Construction: Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS. 2009. Pp. 109 - 125.
12. Mark A. Weiss. Data structures and algorithm analysis in C++. Prentice Hall 3 edition, 2006. Pp. 586.
13. Воеводин Вл.В., Жуматий С.А., Воеводин Вад.В. Практика суперкомпьютера "Ломоносов" // Открытые системы. - Москва: Издательский дом "Открытые системы", 2012. С. 7.