УДК 004.021:004.75

АНАЛИЗ БЫСТРОДЕЙСТВИЯ ПРИ РЕАЛИЗАЦИИ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ ДЛЯ РЕШЕНИЯ ЗАДАЧ ОПТИМИЗАЦИИ В СИСТЕМЕ ЛОГИЧЕСКОГО ПРОЕКТИРОВАНИЯ

И.П. Логинова
Старший научный сотрудник ОИПИ НАНБ,
кандидат технических наук, доцент
irilog@mail.ru

И.П. Логинова

Окончила Белорусский государственный университет, кандидат технических наук, доцент. Работает в ОИПИ НАН Беларуси в должности старшего научного сотрудника и в Белорусском государственном университете информатики и радиоэлектроники в должности доцента. Круг научных интересов: программирование, логическое проектирование и верификация цифровых схем, виртуализация, реализация параллельных алгоритмов.

Аннотация. В работе рассматривается возможность реализации параллельных вычислений в системе FLC-2, предназначенной для технологически независимой оптимизации функциональных и структурных описаний логических схем. На примере работы некоторых программ логической оптимизации демонстрируется методика по организации параллельной работы, в том числе и тех программ, которые представлены исполняемым кодом. Проведены сравнения быстродействия различных вариантов запуска параллельной работы программ оптимизации на потоке известных примеров. Приводятся результаты вычислительных экспериментов на многоядерной системе с общей памятью, подтверждающие эффективность использованного подхода при организации параллельных вычислений в системах логического проектирования для решения трудоемких задач.

Ключевые слова: многоядерный процессор с общей памятью, параллельные процесс, планировщик задач, ускорение, дизъюнктивная нормальная форма (ДНФ), логическая оптимизация, СБИС.

Введение.

В настоящее время решение сложных вычислительных задач проводится на многопроцессорных и многоядерных вычислительных системах, что предусматривает разработку параллельных алгоритмов, учитывающих архитектуру вычислительных систем. При этом основной подход состоит в создании новых алгоритмов или переработке существующих последовательных алгоритмов. Последнее хорошо работает, когда задача может быть разделена по данным. Осуществить распараллеливание комбинаторных задач гораздо сложнее, поскольку для таких задач, как правило, невозможно реализовать одновременное выполнение различных частей алгоритма в одной программе или провести разделение исходных данных. Практически все задачи из области логического проектирования сверхбольших интегральных схем (СБИС), в частности, задачи логической оптимизации, носят комбинаторный характер и имеют экспоненциальную сложность. Особенностью логико-комбинаторных задач является то, что основными объектами преобразований для них являются такие структуры, как булевы переменные, булевы векторы, булевы и троичные матрицы, графы [1-3]. Для задач этого класса характерно то, что объемы перерабатываемой информации при поиске решений относительно невелики, но сами процессы переработки зачастую сопряжены с необходимостью полного перебора и анализа значительного числа вариантов промежуточных решений. Поэтому становится проблематично решать такие задачи большой размерности за приемлемое время.

Актуальность.

Тривиальные параллельные алгоритмы для задач проектирования СБИС, в которых возможно распараллеливание на нужное количество процессоров, описаны в литературе [4-7]. Так, в книге [4], описаны параллельные алгоритмы для размещения и трассировки, проверки соблюдения правил проектирования, логического синтеза, генерации тестов, моделирования ошибок в схемах, моделирования на поведенческом уровне. Исследования, представленные в работах [5-7] показали, что задачи в данной проблемной области, в первых, не допускают автоматического распараллеливания, во-вторых, распараллеливание посредством преобразования последовательных комбинаторно-логических алгоритмов не позволяет получить эффективный параллельный алгоритм. Поэтому разработка параллельных комбинаторно-логических алгоритмов в этой сфере является деятельностью, в которой успехи довольно редки. Тем не менее, возможности использования в САПР высокопроизводительных вычислений интенсивно исследуются из-за важности практических применений. В данной статье описывается подход, который позволяет реализовать параллельные вычисления для многих проектных процедур в САПР СБИС. Организация параллельных вычислений на многоядерном компьютере с общей памятью иллюстрируется на примере программ оптимизации, используемых в процессе проектирования цифровых схем и входящих в состав программного комплекса FLC-2, разработанного в лаборатории логического проектирования ОИПИ НАН Беларуси [9,10]. Реализацию параллельного запуска программ предлагается организовать с использованием технологии OpenMP. В работе проводится оценка быстродействия следующих программ минимизации, работа которых огранизована в параллельном режиме: отечественной программы минимизации булевых функций в классе ДНФ, входящей в систему FLC-2 [10] и зарубежных, свободно распространяемых программ Espresso IIC [12] и ABC [13]. Предложенная методика распараллеливания, основана на имеющейся в FLC-2 возможности разделения исходных данных проектными процедурами. Её использование приводит к сокращению времени выполнения проектных процедур, для которых применима что в итоге позволяет увеличить размерности решаемых задач логического проектирования.

Описание используемой программной среды и объектов для экспериментов.

В работах [9, 10] приведено описание системы логической оптимизации функционально-структурных описаний цифровых устройств FLC-2. В качестве исходных данных для системы выступает функциональное либо иерархическое функционально-структурное описание проектируемой цифровой схемы. Результатом является оптимизированное описание, на основе которого можно получить логическую схему меньшей сложности и большего быстродействия. В настоящее время существенно расширился инструментарий системы и возможности базового функционального обеспечения, частности, в системе FLC-2 проектные процедуры позволяют проводить оптимизацию исходных описаний используя разнообразные методы, как на основе структур данных разработанных для FLC-2, так и с привлечением программ логической оптимизации, имеющихся в свободном доступе. В качестве языка описания логических схем в FLC-2 используется язык SF [9], который ориентирован на иерархическое описание логической схемы. Иерархия описания схемы представляется в виде дерева, вершине дерева соответствует отдельный блок. Любой не листовой блок, соответствующий узлу дерева иерархии, выражается заданием связей входящих в него подсхем. Описание схемы на языке SF представлено последовательностью функционально-структурных описаний подсхем (блоков). Функциональные описания листовых блоков дерева иерархии представляют собой либо логические уравнения – скобочные выражения в булевом базисе И, ИЛИ, НЕ (так называемый LOG-формат), либо матричные формы представления систем булевых функций в виде ДНФ (так называемый SDF-формат). Система ДНФ булевых функций задается парой матриц: строки троичной матрицы задают элементарные конъюнкции, входящие в систему ДНФ, в булевой матрице единицы задают вхождения конъюнкций в ДНФ функций. Если головной блок описан на функциональном уровне, то в этом случае весь проект описания схемы представляет собой один листовой блок. Форматные преобразования в FLC-2 позволяют выполнять преобразование многоуровневых представлений в виде уравнений из LOG-формата в матричную форму системы ДНФ (SDF-формат). Таблицы истинности систем булевых функций также представляются в матричной форме (SDF-формат). Методы разбиения логической схемы на отдельные подсхемы (блоки)

реализуемые посредством применения соответствующих проектных процедур FLC-2, преследуют различные цели. Для всей схемы и каждого из блоков, в отдельности, могут быть применены методы оптимизации, в том числе методы минимизации в классе ДНФ. Возможность работы с матричными формами и иерархическими описаниями является важной отличительной особенностью языка SF и системы FLC-2 в целом. Конвертация в объекты, представленные на языке описания аппаратуры интегральных схем [14] (и обратно) используется для импорта данных в FLC-2, а также для экспорта оптимизированных SF-описаний с целью их моделирования и схемной реализации. Одним из ключевых моментом при проведении преобразований в рамках программного комплекса FLC-2 является верификация проектных решений на всех этапах маршрута [15,16]. Программы верификации работают для любых пар проектных состояний, как на уровне SF-описаний, так и на уровне объектов, представленных на языке описания аппаратуры.

Программы логической оптимизации, используемые в экспериментах.

Логическая минимизация в классе ДНФ [1, 17-19] реализует алгоритмы совместной и раздельной минимизации по различным критериям (числу элементарных конъюнкций, суммарному числу литералов в конъюнкциях и др.). Ниже перечислены программы, с которыми проводились эксперименты по организации параллельных вычислений.

Программа Espresso [12] является самой известной программой минимизации и предназначена для совместной и раздельной минимизации систем полностью определенных и частичных булевых функций (и систем многозначных функций) в классе ДНФ по различным критериям – числу элементарных конъюнкций и числу литералов. Существуют варианты и модификации данной программы. В экспериментах, результаты которых приводятся далее, использовалась программа Espresso IIC (далее – Espresso) совместной минимизации системы полностью определенных функций.

Программа ABC основана на графовых моделях многоуровневых BDD-представлений систем функций. В программе использованы специальные структуры данных, разработанные для решения этой задачи. Алгоритмы ABC эффективно работают для раздельной минимизации ДНФ булевых функций большой размерности [13].

Программа Minim [17] включена в систему FLC-2, предназначена для совместной и раздельной минимизации систем как полностью определенных, так и частичных булевых функций, заданных в интервальной форме, использует в качестве входных и выходных данных описание логической схемы в формате SDF [9]. В проведенных экспериментах программа Minim использовалась для выполнения раздельной минимизации полностью определенных булевых функций, в которой критериями минимизации являются число конъюнкций в кратчайшей ДНФ каждой компонентной функции. Программа реализует обобщение алгоритма [18], использует специальные структуры данных для векторно-матричных объектов (булевых и троичных векторов, матриц) [2,3]. При проведении раздельной минимизации программой Minim использовался итерационный алгоритм, ориентированный на получение решения лучшего качества.

Примеры систем функций, используемые в экспериментах.

В работе [11] были представлены результаты экспериментов с программами минимизации систем булевых функций в классе дизъюнктивных нормальных форм, а также проведен подробный анализ эффективности и качества результатов оптимизации. Эксперименты в исследовании [11] проводились на четырех наборах систем булевых функций. Для экспериментов, представленных в этой работе, были отобраны 25 примеров из комплекса примеров [11]. В таблице 1 приведены основные параметры примеров, с которыми проводились эксперименты. Первую группу примеров (1-14) составляют системы функций, описывающие функционирование устройств перемножения и устройств возведения в квадрат целых неотрицательных чисел. Подобные примеры систем функций появляются, при решении задач перепроектирования логических схем из одного базиса логических элементов в другой. Вторую группу примеров (15-19) составляют системы булевых функций, взятые из практики проектирования управляющей логики заказных СБИС: примеры S12, …, S16 – таблицы истинности, задающие SF-описания систем некоторых функций: пример 25 – система ДНФ, которая задает блок управляющей

логики в заказной СБИС. Третья группа примеров (20-24) сформирована из псевдослучайных систем ДНФ [11].

Таблица 1. Параметры систем ДНФ для экспериментов

Исходные данные – моноблок (SF-описание, моноблок)					*Исходные данные – сеть (SF-описание, 2Connect- формат)*		
№ п/п	Имя примера	*n*	*m*	*k*	Число блоков	Размер файла (Мб)	$\sum_{i=1}^{m} k_i$
Первый набор примеров							
1	square_12_DNF	12	24	5 612	24	0,214	5 902
2	square_12_TABL	12	24	4 096	24	0,156	39 962
3	square_13_DNF	12	26	11 259	26	0,462	11 904
4	square_13_TABL	14	26	8 191	26	0,336	87 928
5	square_14_DNF	14	28	22 867	28	0,98	24 151
6	square_14_TABL	15	28	16 384	28	0,720	192 100
7	square_15_DNF	15	30	45 568	30	2,08	48 045
8	square_15_TABL	14	30	32 768	30	1,50	416 811 2
9	mult_7_DNF	14	14	13 060	14	0,395	13 252
10	mult_7_TABL	16	14	16 384	14	0,496	93 334
11	mult_8_DNF	16	16	52 810	16	1,76	53 621
12	mult_8_TABL	18	16	65 536	16	2,18	434 660
13	mult_9_DNF	18	18	208 598	18	7,75	212 047
14	mult_9_TABL	18	18	262 144	18	9,75	1 302 835
Второй набор примеров							
15	S12	13	12	4 096	13	12	27 238
16	S13	14	13	8 192	14	13	58 616
17	S14	15	14	16 384	15	14	125 497
18	S15	16	15	32 768	16	15	267 551
19	S16	16	16	65 536	16	16	568 043
Третий набор примеров							
20	Psevdo1	15	20	1 000	20	0,0315	2 103
21	Psevdo2	20	20	5 000	20	0,180	10 676
22	Psevdo3	25	20	10 000	20	0,410	26 657
23	Psevdo4	30	20	25 000	20	1,12	92 300
24	Psevdo5	25	20	30 000	20	1,48	150 620
25	Sistem8	25	20	45 548	20	2,08	45 947

Последовательным применением ряда проектных процедур и форматных преобразований в системе FLC-2 по исходным описаниям всех примеров из таблицы 1 получено разбиение каждого исходного описания, представленного в виде одной ДНФ (далее моноблока), в описания логической сети (формат 2-CONNECT). В таблице 1 приведены параметры логических сетей, блоками которых являются однокомпонентные функции в виде ДНФ. В предпоследнем и последнем столбцах таблицы 1 для примеров 1-25 дополнительно приведена следующая информация: размер файла сети (Мб) и общее число элементарных конъюнкций в блоках логической сети.

Описание предлагаемых подходов и алгоритмов.

Далее рассматриваются нескольких подходов по организации параллельных вычислений с участием упомянутых выше программ минимизации. В работе [20] представлена модификация алгоритма раздельной минимизации программы Minim, в котором параллельное выполнение реализовано путем использования директив OpenMP непосредственно в коде программы Minim. Также в [20] приведен фрагмент алгоритма программы раздельной минимизации, в котором показана организация этого подхода. Псевдокод этого фрагмента программы представлен на рисунке 1.

```
MiRi(Limit, Crit, CSOP &Sop1, CSOP & ResSop)
{
  n=Sop1.foo1(), m=Sop1.foo2(), k=Sop1.foo3(); // n, m, k – входы/выходы/конъюнкции
  CSOP SopR(….); // создать объект для ДНФ
  num_core=omp_get_num_procs ();               // определить число ядер
#pragma omp parallel for shared(Limit, Crit, Sop1, SopR, n, m) num_threads(num_core)
  for (i = 0 ; i < m ; i++)
  {
#pragma omp task firstprivate(i) shared(Sop1, k, SopR, n, m)
   {
     CTM Tm, Tm1;    // создать два объекта: Tm, Tm1 – троичные матрицы
     CSOP SopRR(….); // в каждой task объект для ДНФ
......
     Tm1=foo4(…..);
......
     if(Tm1.foo5())
     {
       MiOi (Limit, Crit, Tm1, Tm); // минимизация одной полностью определенной функции
       for (j = 0; j < Tm.foo4(); j++)
       { SopRR.foo5(Tm.foo6(j),….); }
 #pragma omp critical
         {
           SopR.foo7(…SopRR);
         } // конец критической секции
       }   // конец проверки условия в 15-й строке
     }     // конец кода в блоке task
  }        // конец кода в блоке parallel
  ResSop.foo8(…SopR); // результирующая система ДНФ
}
```

Рисунок 1. Псевдокод параллельного алгоритма раздельной минимизации (MinimPar)

Другой подход заключается в использовании директив OpenMP для организации параллельной работы группы процессов, каждый из которых запускает программу минимизации со своим набором исходных данных. Такие условия в полной мере соответствуют проведению минимизации для блоков, входящих в состав логической сети, если предварительно провести разделение логической сети на файлы с отдельными блоками. Поэтому в проводимых экспериментах перед параллельным запуском процессов для каждого примера из таблицы 1 проводится предварительное разделение сети на отдельные блоки.

Параллельный алгоритм раздельной минимизации системы булевых функций.

Как было показано в [20] распараллеливание алгоритма раздельной минимизации, когда каждая из функций системы минимизируется независимо от других, может быть сравнительно просто реализовано с использованием директив OpenMP. Объектом распараллеливания является программный цикл (по числу компонентных функций), в котором выполняется минимизация для одной функции. Нативный алгоритм минимизации однокомпонентной функции впервые описан в работе [17]. В экспериментах, результаты которых представлены ниже, используется одна из разновидностей этого алгоритма – т.н. интерационный метод. Как было упомянуто выше, на рисунке 1 представлен псевдокод параллельного алгоритма раздельной минимизации. Описание структур данных для участвующих в алгоритме объектов, а также назначение фрагментов кода более подробно рассмотрены в [20]. В алгоритме задействован механизм так называемых задач, который определяется заданием в коде директив OpenMP – **task** и **parallel for**. Эти директивы проводят с отдельные фрагментами кода следующие действия. После OpenMP-директивы **parallel for** образуются параллельные регионы. В конце параллельного региона группы потоков останавливаются, а выполнение основного потока продолжается. Далее внутри отдельного параллельного региона посредством OpenMP-прагмы **task** в текущем потоке выделяется отдельная независимая задача, с которой ассоциирован соответствующий блок операторов. Такая задача помещается внутрь параллельной области и может выполняться в отдельном потоке. Однако задача **task** не образует новый поток, а помещается в пул, из которого ее может взять один из свободных потоков. Задача может выполняться немедленно после создания или быть отложенной на неопределенное время и выполняться по частям. Размер таких частей определяется размером фрагмента операторов, а порядок выполнения частей текущих или отложенных задач определяется системным планировщиком задач и условиями синхронизации. Динамическое распределение задач по потокам осуществляется алгоритмами планировщика задач [20]. С помощью OpenMP-директивы **critical** оформляется критическая секция

программы, в которой собираются минимизированные ДНФ компонентной функции, полученные отдельными задачами. В каждый момент времени в критической секции может находиться не более одной задачи. Все другие задачи будут заблокированы, пока вошедшая задача не закончит выполнение. Как только вошедший в критическую секцию поток выйдет из нее, один из заблокированных потоков туда войдет. Если на входе стоит несколько потоков, то случайным образом выбирается один из них, а остальные продолжают ожидание. Естественно, введение критических секций снижает степень параллелизма. Задачи **task** являются альтернативой традиционным потокам **threads**. Механизм задач позволяет реализовать лучший баланс нагрузки, эффективное использование имеющихся ресурсов и предлагает высокий уровень абстракции, благодаря которому можно не заботиться о непосредственном управлении параллельным исполнением. Время нахождения задачи в критической секции, где формируется объединение ДНФ текущих решений, определяется временем нахождения минимальной ДНФ для одной компонентной функции. Модификацией программы Minim, которая проводит параллельные вычисления согласно алгоритму на рисунке 1 является программа MinimPar. Исходными данными для программы MinimPar служит система ДНФ, представленная одним блоком в формате SDF.

Алгоритм запуска параллельных Qt-процессов.

В основу алгоритмов, которые реализируют параллельную работу программ минимизации, положено создание и использование процесса. Процесс – это экземпляр программы, загруженной в память компьютера для выполнения. Каждый процесс имеет свою область памяти. Когда программа производит запуск другой программы, средства ОС всегда создает новый процесс (дочерний). В среде разработки Qt [21] можно создавать процессы с помощью класса Qprocess, который используется для запуска внешних программ и связи с ними, содержит методы для манипулирования системными переменными процесса. Работа с объектами класса QProcess может осуществляться в синхронном и асинхронном режимах. Запуск процесса выполняется методом **start()**, в который необходимо передать имя команды и список ее аргументов. В поле аргументов может быть введена любая команда, в том числе это может быть командная строка, запускающая внешнюю программу. Фрагмент программы StartParBlock, псевдокод которого приведен на рисунке 2, организует запуск в параллельном режиме процессов с исполняемыми модулями запускаемых программ. На рисунке 2 показано, что для каждого процесса формируются командные строки для запуска программ минимизации, участвующих в экспериментах: Minim, MinimPar, ABC и Espresso. Входными данными являются блоки, полученные в результате разбиения логической сети. Каждый блок содержит описание ДНФ однокомпонентной функции.

```
// n , m – число входов, число выходов (блоков), num_core – число ядер
#pragma omp parallel for firstprivate(…)shared(…,NameProg,m) num_threads(num_core)
 for(i=1; i<=m; i++) // цикл по числу блоков (m)
 {
#pragma omp task untied firstprivate(i, …) shared (…,NameProg,m)
  { ……
    QProcess* m_process = new QProcess(); //объект –новый Qt-процесс
    m_process->setProcessChannelMode(QProcess::MergedChannels);
    // образование канала для вывода протоколов i-го процесса
    m_process->setStandardOutputFile(("Qt_protocol.log");
    QString strCommand = NameProg;  // занести в ком. строку имя программы NameProg
    QString strCommand1; // аргументы ком. строки для запуска программы
    strCommand1+="-i"+W_in.data();  // исходный sf-файл i-го блока
    strCommand1+="-o"+W_out.data(); // sf-файл результата минимизации i-го блока
    strCommand+=strCommand1;
     // Формирование аргументов для программ минимизации
    if(NameProg=="Minim")           // формирование аргументов для программы Minim
       strCommand +="-r"+ config.ini";
    if(NameProg=="MinimPar")        // формирование аргументов для программы MinimPar
       strCommand +="-r"+ config.ini";
    if(NameProg=="Espresso")          // формирование аргументов для программы Espresso
       strCommand +="-f"+"Probe.prb"; // дополнительные параметры для Espresso
    if(NameProg=="ABC")              // формирование аргументов для программы ABC
       strCommand +=" -r "+Name_config; // для каждого блока свой ini-файл
    m_process->start(strCommand);       // запуск i-го процесса
    m_process->waitForFinished(-1);     // режим i-го процесса - ассинхронный
    rc = m_process->exitCode();
  } // конец omp-директивы task
}   // конец цикла по числу блоков (Qt-процессов)
foo1(……); // Собрать время всех Qt-процессов в один протокол
foo2(……); // Собрать оптимизированные sf-описания в одну логическую сеть
```

Рисунок 2. Псевдокод алгоритма параллельного запуска Qt-процессов (StartParBlock)

Для работы с программами Espresso и ABC описания блоков (формат SDF) преобразуются во входные/выходные описания, используемые этими программами. Такие преобразования проводятся программами конвертирования, предоставляемыми системой FLC-2. В каждом процессе запускается и выполняется программа минимизации для отдельного блока, начиная с чтения исходных данных блока и завершая записью минимизированного описания блока. После размещения прагмы task создается пул задач, каждой задаче назначен свой процесс [20]. В программе StartParBlock организован замер времени начала и окончания каждой задачи, сохранение этих значений в протоколе, что позволяет определить порядок и длительность выполнения каждой задачи. В качестве иллюстрации на рисунке 3 показано распределение процессов по ядрам при параллельном запуске программы MinimPar для блоков примера 22. После обработки полученного протокола определены длительность и порядок выполнения каждой задачи, По этим данным построена интервальная диаграмма, которая показывает, что планировщик задач направил ожидавшие своей очереди процессы-задачи из пула на ядра, которые быстрее всего завершили выполнение процесса.

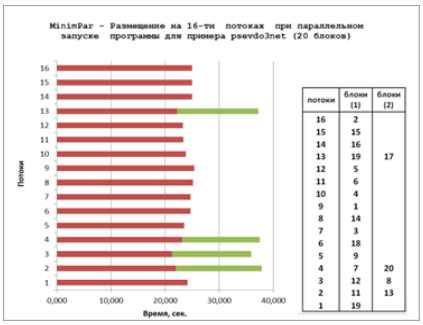

Рисунок 3. Параллельный запуск Qt-процессов. Распределение 20-ти процессов на 16 ядер

Алгоритм параллельного запуска Spawn-процессов.

Системные вызовы среды выполнения Майкрософт также позволяют создавать и исполнять процессы. Системный вызов работает подобно вызову подпрограммы. На рисунке 4 приведен псевдокод алгоритма параллельного запуска с использованием Spawn-процессов. Каждая из запускаемых в цикле системных функций **_spawnlp** создает и запускает новый процесс, который, как и в случае Qt-процесса, можно определить как асинхронный.

```
// n ,m – число входов, число выходов, num_core – число ядер
#pragma omp parallel for firstprivate(…) shared(…,NameProg, m) num_threads(num_core)
 for(int i=1; i<=m; i++) // запуск spawnlp процессов в цикле
 {
#pragma omp task firstprivate(…,i) shared(… NameProg, m)
  {
    if(NameProg=="ABC") // запуск программы ABC
    {
      W_ini=" -r ";
      W_ini+=Name_config.c_str(); // для каждого блока был создан свой config.ini
      result=spawnlp(…,NameProg,W_in,W_out,W_ini, NULL);
    }
    if(NameProg=="Espresso")      // запуск программы Espresso
    {
      W_ini=" -f ";
      W_ini+=TargetPrb.c_str();   // для каждого блока свой config.ini и TargetPrb
      result=spawnlp(…,NameProg,W_in,W_out,W_ini, NULL);
    }
    if(NameProg=="MinimPar")      // запуск программы MinimPar или Minim
    {
      result=spawnlp(…, NamePrg, W_in, W_out, NULL);
    }
  }   // конец task
}     // конец цикла for для всех spawnlp-processes
foo1(……); // Собрать время всех spawnlp-processes в один протокол
foo2(……); // Собрать оптимизированные sf-описания в одну логическую сеть
```

Рисунок 4. Псевдокод алгоритма параллельного запуска spawn-процессов (StartParBlock)

Описание экспериментов.

Цель экспериментов: получение временных характеристик параллельных вычислений, анализ быстродействия предложенных алгоритмов организации параллельных вычислений с программами минимизации, которые используются в САПР логического проектирования.

Эксперимент 1. Сравнение времени работы программ Minim и MinimPar.

Программы Minim и MinimPar выполнялись с установкой следующих опций (раздельная минимизация, представление системы ДНФ булевых функций в виде моноблока и 2Connect-сети). В данном эксперименте проводится сравнение временных характеристик для программ Minim и MinimPar для двух представлений исходных данных. В первом разделе таблицы 2 для моноблока организован запуск этих программ в последовательном и параллельном режимах, указаны времена выполнения программ Minim и MinimPar, вычислены коэффициенты ускорения для каждого примера. В примерах 13 и 14 для программы Minim (моноблок, последовательное) указана пометка «нет решения», которая означает, что программой Minim задача не решена за приемлемое время (более 8 часов).

Таблица 2. Время выполнения Minim, MiniPar (моноблок), MinimPar (сеть)

Программа Minim – раздельная минимизация, итерационный метод									
Моноблок (SF-описание, формат SDF)				2Connect сеть(SF-описание, формат SDF)					
№ п/п	Minim последов.	MinimPar паралл.	ускор.	Запуск MinimPar, Qt-процесс			Запуск MinimPar, Spawn-процесс		
				последов.	паралл.	ускор.	последов.	паралл.	ускор.
	время, с	время, с		время, с	время, с		время, с	время, с	
1	16,978	5,043	3,4	16,439	4,914	3,3	19,549	7,55	2,6
2	18,747	5,435	3,4	18,212	5,352	3,4	18,048	5,311	3,4
3	119,625	40,617	2,9	119,758	40,048	3,0	70,817	15,108	4,7
4	97,556	29,031	3,4	97,55	29,702	3,3	74,134	14,944	5,0
5	754,733	259,493	2,9	750,31	256,85	2,9	139,963	29,131	4,8
6	622,005	177,737	3,5	604,298	182,075	3,3	152,655	33,643	4,5
7	2223,512	416,663	5,3	2204,563	413,596	5,3	449,173	125,801	3,6
8	2305,956	421,526	5,5	2286,564	431,915	5,3	533,564	142,666	3,7
9	344,828	130,368	2,6	335,795	130,147	2,6	89,633	20,953	4,3
10	343,124	143,878	2,4	331,32	143,485	2,3	100,427	21,319	4,7
11	2429,447	562,579	4,3	2390,28	543,863	4,4	1024,369	367,759	2,8
12	2964,118	560,962	5,3	2670,287	560,198	4,8	1324,489	417,062	3,2
13	–	7119,39	–	19589,14	7045,246	2,8	22671,134	6919,568	3,3
14	–	7857,82	–	25765,78	7695,867	3,3	25346,841	7637,743	3,3
15	32,725	9,555	3,4	30,541	9,709	3,1	29,843	9,719	3,1
16	176,188	63,125	2,8	175,293	63,793	2,7	69,64	14,435	4,8
17	763,559	262,684	2,9	756,968	264,407	2,9	149,959	34,445	4,4
18	2043,163	218,962	9,3	2020,064	426,186	4,7	549,73	153,355	3,6
19	2429,447	708,848	3,4	3587,953	711,828	5,0	2351,599	667,248	3,5
20	0,676	0,151	4,5	1,249	0,237	5,3	0,898	0,224	4,0
21	43,168	9,114	4,7	44,412	9,669	4,6	42,847	9,174	4,7
22	235,770	48,145	4,9	236,041	45,842	5,1	235,558	63,728	3,7
23	246,861	48,724	5,1	339,52	54,317	6,3	268,335	42,677	6,3
24	844,687	163,83	5,2	928,028	183,873	5,0	610,607	108,797	5,6
25	3503,525	644,533	5,4	2778,772	639,848	4,3	1366,052	441,168	3,1

После анализа результатов эксперимента, представленных в таблице 2, можно сделать следующие выводы. Размещение директив OpenMP **parallel for** и **task** в коде программы MinimPar, дает сопоставимые

временные характеристики с характеристиками, полученными при запуске параллельных процессов с программами Minim и MinimPar для сети, что демонстрирует график на рисунке 5. График на рисунке 6 иллюстрирует значения ускорения для вариантов распараллеливания, реализованного с участием программы MinimPar. На рисунке 6 в помеченный эллипсом диапазоне значения ускорения для примеров 13 и 14 не определены, так как в таблице 2 не представлено время выполнения минимизации в последовательном режиме для этих двух примеров.

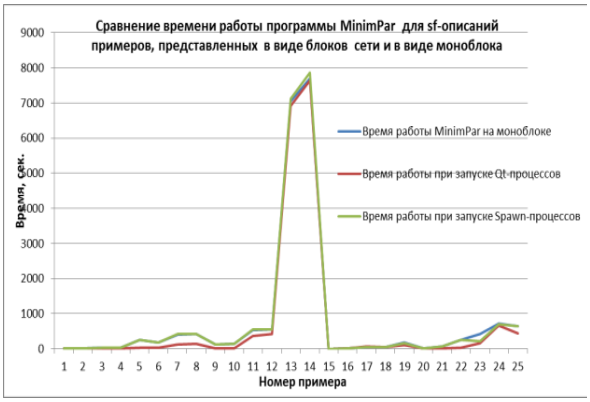

Рисунок 5. Время выполнения параллельных вычислений с MinimPar

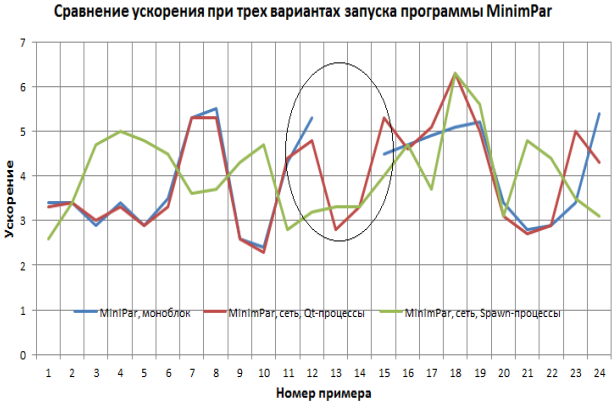

Рисунок 6. Ускорение при параллельных вычислениях с MinimPar

Анализ графиков на рисунках 5 и 6 показывает, что 1) при задании исходных данных в формате моноблока время выполнения MinimPar близко к значениям времени выполнения этой программы, запускаемой с использованием параллельных процессов; 2) график 6 иллюстрирует сопоставимые значения ускорения для программы MinimPar при работе на моноблоке и на сети (Qt-процессы); 3) График 6 показывает, что при запуске параллельных Spawn-процессов получены другие значения ускорения. В целом, разброс значений ускорения от 3 до 6 обусловлено параметрами примеров: размерами файлов и числом блоков. Вывод по результатам этого эксперимента следующий – MinimPar, а также параллельный запуск процессов с этой программой дает увеличение быстродействия в среднем в 4 раза.

Эксперимент 2. Оценка времени работы программы Espresso – два механизма запуска процессов.

Выполнение Espresso осуществляется посредством запуска последовательных и параллельных процессов. Проводится сравнение временных характеристик, полученных при использовании двух механизмов запуска процессов – Qt- и Spawn- процессов. Для Espresso исходными данными являются конвертированные описания блоков сетей примеров 1-25. В таблице 3 указаны: время минимизации программой Espresso для каждой сети при запуске процессов двух типов, вычислены значения ускорения. На рисунке 7 приведены графики, которые построены согласно временным характеристикам, полученным при запуске последовательных и параллельных процессов с программой Espresso. На рисунке 8 показаны графики ускорения для всех вариантов распараллеливания, реализованных с участием Espresso. Графики показывают практически схожие временные характеристики, полученные при минимизации сетей программой Espresso. По результатам этих экспериментов можно сделать следующий вывод, что параллельный запуск с использованием программы Espresso для всех примеров не зависит от вида процессов и показывает сопоставимые характеристики.

Эксперимент 3. Оценка времени работы программы ABC – два механизма запуска процессов.

Программа ABC запускается последовательными и параллельными процессами двух типов: Qt-, Spawn- процессы. Для ABC также проведено конвертирование описаний блоков сетей для примеров 1-25. В таблице 4 указаны времена выполнения минимизации программой ABC, вычислены значение ускорения.

Таблица 3. Время выполнения Espresso (сеть), Qt- и Spawn- процессы

Программа минимизации Espresso						
№ п/п	время, с., Qt-процесс, последов.	время, с., Spawn-процесс, последов.	время, с., Qt-процесс, параллельн.	время, с., Spawn-процесс, параллельн.	ускорение, Qt-процесс	ускорение, Spawn-процесс
1	1,893	1,68	0,4	0,328	4,7	5,1
2	2,105	1,88	0,454	0,42	4,6	4,5
3	2,856	2,597	0,618	0,584	4,6	4,4
4	3,629	3,414	0,862	0,786	4,2	4,3
5	6,224	5,96	1,545	1,348	4,0	4,4
6	8,568	8,491	1,996	1,829	4,3	4,6
7	17,964	17,452	4,712	4,558	3,8	3,8
8	28,335	28,096	6,739	6,325	4,2	4,4
9	3,297	3,197	0,896	0,853	3,7	3,7
10	5,03	4,894	1,223	1,125	4,1	4,4
11	51,134	50,899	15,292	15,223	3,3	3,3
12	85,318	85,913	21,227	21,039	4,0	4,1
13	1862,56	1868,36	692,084	700,964	2,7	2,7
14	2867,787	2853,32	898,208	894,552	3,2	3,2
15	1,336	1,176	0,337	0,272	4,0	4,3
16	2,982	2,8	0,622	0,563	4,8	5,0
17	21,027	20,866	3,895	3,829	5,4	5,4
18	97,641	97,057	21,294	19,982	4,6	4,9
19	462,743	463,262	106,091	105,88	4,4	4,4
20	1,224	1,129	0,295	0,282	4,1	4,0
21	2,294	2,17	0,533	0,472	4,3	4,6
22	6,569	6,445	1,595	1,452	4,1	4,4
23	24,028	23,883	6,019	5,486	4,0	4,4
24	97,262	97,802	22,332	21,653	4,4	4,5
25	91,111	89,757	28,445	28,126	3,2	3,2

Рисунок 7. Время выполнения программы минимизации Espresso

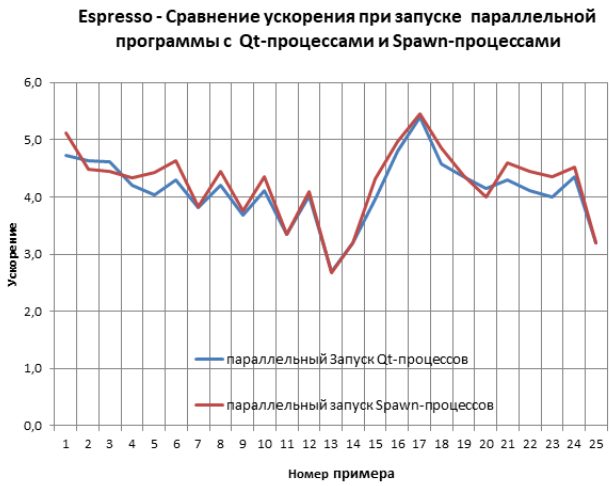

Рисунок 8. Ускорение при параллельных вычислениях с Espresso – два типа процессов

Таблица 4. Время выполнения ABC (сеть), Qt- и Spawn- процессы

Программа ABC – раздельная минимизация						
№ п/п	время, с., Qt-процесс, последов.	время, с., Spawn-процесс, последов.	время, с., Qt-процесс, параллельн.	время, с., Spawn-процесс, параллельн.	ускорение, Qt-процесс	ускорение, Spawn-процесс
1	3,33	4,877	0,548	0,988	4,9	6,1
2	2,673	5,658	0,556	0,945	6,0	4,8
3	3,07	5,449	0,61	1,096	5,0	5,0
4	3,324	6,125	0,647	1,176	5,2	5,1
5	3,457	6,351	0,656	1,391	4,6	5,3
6	4,022	7,719	0,771	1,43	5,4	5,2
7	4,298	8,179	0,878	1,639	5,0	4,9
8	5,09	9,802	1,087	1,917	5,1	4,7
9	1,79	3,309	0,376	0,669	4,9	4,8
10	2,005	3,808	0,388	0,759	5,0	5,2
11	3,127	5,997	0,761	1,394	4,3	4,1
12	3,912	7,67	0,887	1,654	4,6	4,4
13	14,286	28,48	4,359	8,589	3,3	3,3
14	12,873	25,658	3,059	6,209	4,1	4,2
15	2,276	4,097	0,547	0,85	4,8	4,2
16	3,425	6,411	0,719	1,277	5,0	4,8
17	13,464	26,496	2,659	5,458	4,9	5,1
18	28,61	56,9	5,919	11,121	5,1	4,8
19	123,284	245,824	34,06	69,565	3,5	3,6
20	1,524	2,74	0,32	0,535	5,1	4,8
21	1,796	3,332	0,386	0,65	5,1	4,7
22	2,266	4,369	0,455	0,889	4,9	5,0
23	3,124	6,256	0,682	1,267	4,9	4,6
24	4,922	9,677	1,081	2,075	4,7	4,6
25	3,885	7,309	0,845	1,628	4,5	4,6

На рисунке 10 представлены графики, отображающие временные характеристики, полученные при запуске как последовательных, так и параллельных процессов.

На рисунке 11 приведены графики, на котором представлены значения ускорения при параллельных вычислениях с программой ABC для двух видов процессов.

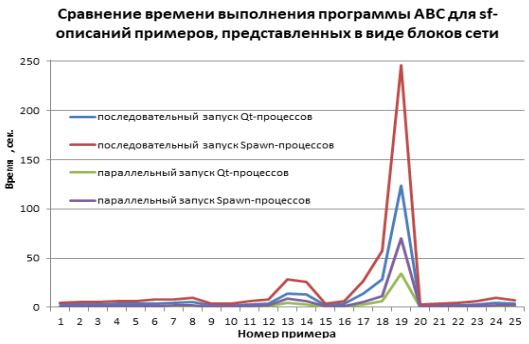

Рисунок 10. Время выполнения программы минимизации ABC

Рисунок 11. Ускорение при параллельных вычислениях с ABC – два типа процессов

Графики на рисунке 10 четко демонстрируют, что временные характеристики минимизации сети программой ABC при использовании Qt-процессов в последовательном и параллельном режимах для всех примеров эффективнее, чем временные характеристики минимизации программой ABC при использовании запуска Spawn-процессов в последовательном и параллельном режимах. График на рисунке 11 показывает, что для всех примеров значения ускорения для Qt- и Spawn-процессов близки. График на рисунке 12 показывает соотношение времени выполнения программы ABC для Qt- и Spawn-процессов. Анализ этого графика позволяет сделать вывод, что проведение параллельных вычислений с программой ABC лучше проводить посредством запуска параллельных Qt-процессов.

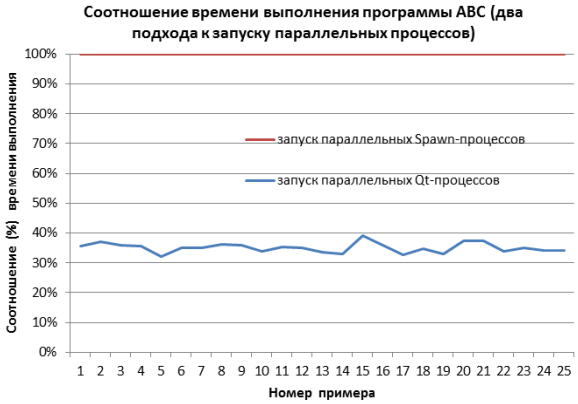

Рисунок 12. Соотношение времени на минимизацию сетей с ABC для Qt- и Spawn-процессов

Эксперимент 4. Организации параллельного запуска для пакета проектов.

В параллельных циклах алгоритмов, приведенных на рисунках 2 и 4, запускаются программы для любого числа блоков. К блокам одного примера можно добавлять блоки, полученные в результате разбиения логической сети другого примера. Программные средства, которые интегрированы с системой FLC-2 и обеспечивают запуск процессов в разных режимах, позволяют проводить объединение блоков нескольких сетей для проведения минимизации одной программой или, наоборот, осуществлять в одном параллельном цикле запуск разных программ, каждая работает со своим пакетом блоков. Как пример, на рисунках 12 и 13 приведены интервальные диаграммы Ганта, построенные после анализа протоколов, полученных при параллельном запуске программ MinimPar и ABC для пакета из блоков примеров 17 и 21.

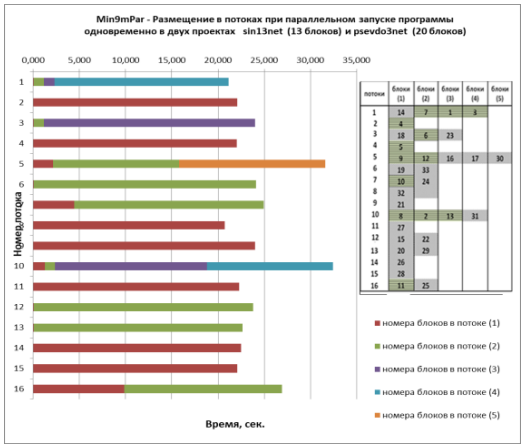

Рисунок 12. Программа MinimPar: распределение задач (процессов) для пакета

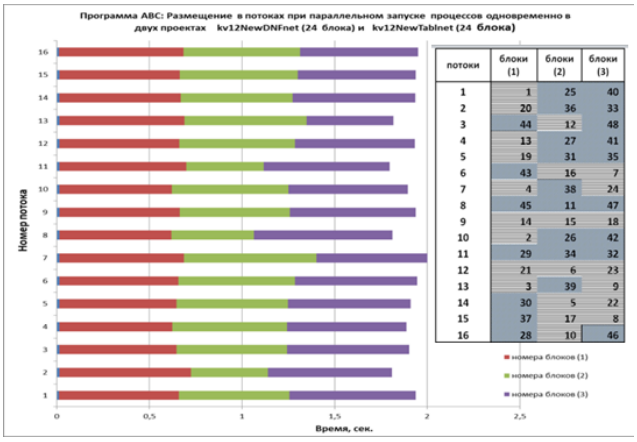

Рисунок 13. Программа ABC: распределение задач (процессов) для пакета

В правой части диаграмм приведены таблицы, в которой показано распределение процессов по ядрам. Из этих диаграмм, видно, что планировщик задач направляет процессы из пула задач вне

зависимости от того, к блокам какого проекта они относятся. В таблице 5 приведены значения ускорения при проведении» параллельного запуска процессов для пакета: в 2 раза для программы MinimPar, в 1,89 раза для программы ABC (эксперимент проводился для пакета из 2-х примеров). Таким образом, объединение в пакет дает дополнительное ускорение к тем значениям, которое получается при параллельной минимизации сети отдельного примера.

Таблица 5. Время выполнения программ MinimPar и ABC при объединении блоков 2-х примеров

Параллельная минимизация объединенных блоков двух логических сетей в пакет							
Программа MinimPar				Программа ABC			
Параллельный запуск Spawn-процессов				Параллельный запуск Spawn-процессов			
№ примера	Имя примера	Число блоков	Время, сек	№ примера	Имя примера	Число блоков	Время, сек
17	S13	14	14,435	17	S13	14	0,65
21	psevdo3	20	63,728	21	psevdo3	20	5,458
17 и 21	S13 и psevdo3	34	37,92	17 и 21	S13 и psevdo3	34	3,232
Ускорение (пакет)		**2,1**		Ускорение (пакет)		**1,89**	

Заключение.

Предложены алгоритмы по организации параллельных вычислений для программ оптимизации в системе логического проектирования. Данный подход показал эффективность по времени по сравнению с работой этих программ в последовательном режиме. По результатам экспериментов на 8-ми ядерном процессоре Intel i7 получено ускорение в среднем, от 3 до 5. Эксперименты также показали, что проведение параллельного выполнения программ минимизации для пакета примеров дает дополнительное ускорение. Рассмотренные в данной работе подходы предлагают альтернативу разработке новых параллельных алгоритмов для решения комбинаторных задач. Кроме того, этот подход позволяет реализовать в системах проектирования параллельные вычисления с использованием «внешних» программ, которые представлены только исполняемыми кодами.

Список литературы

[1] Закревский, А.Д. Логические основы проектирования дискретных устройств / А.Д. Закревский, Ю.В. Поттосин, Л.Д. Черемисинова. – М.: Физматлит, 2007. – 589 с.

[2] Романов, В. И. Булевы векторы и матрицы в C++ / В. И. Романов, И. В. Василькова // Логическое проектирование : сб. науч. тр. – Минск : Ин-т техн. кибернетики НАН Беларуси, 1997. – Вып. 2. – С. 150–158.

[3] Черемисинов, Д. И. Троичные векторы и матрицы в C++ / Д. И. Черемисинов, Л.Д. Черемисинова // Логическое проектирование : сб. науч. тр. – Минск : Ин-т техн. кибернетики НАН Беларуси, 1998. – Вып. 3. – С. 146–156.

[4] Banerjee, P. Parallel Algorithms For VLSI Computer-Aided Design / P.Banerjee – Prentice Hall, Englewoods Cliffs, NJ, 1994. – 699 p.

[5] Черемисинов, Д.И. Проектирование и анализ параллелизма в процессах и программах / Д.И. Черемисинов. – Минск: Беларус. навука, 2011. – 300 с.

[6] Торопов, Н.Р. Параллельные логико-комбинаторные вычисления в среде MPI / Н.Р. Торопов // Информатика. – 2005. – № 3. – С. 82–90.

[7] Hamadi, Y. ManySAT: a Parallel SAT Solver/ Y. Hamadi, S. Jabbour, L. Sais // Journal on Satisfiability, Boolean Modeling and Computation, 2009, Vol. 6, pp. 245–262.

[8] Черемисинов, Д.И. Использование параллельных вычислений при автоматизированном проектировании СБИС / Д.И. Черемисинов, Л.Д. Черемисинова // Проблемы разработки перспективных микро- и наноэлектронных систем – 2016. Сборник трудов / под общ. ред. академика РАН А.Л. Стемпковского. – М.: ИППМ РАН, 2016. Часть – С. 32-39.

[9] Бибило, П. Н. Логическое проектирование дискретных устройств с использованием продукционно-фреймовой модели представления знаний / П.Н. Бибило, В.И. Романов. – Минск: Беларус. навука, 2011. – 279 с.

[10] Бибило, П. Н. Система логической оптимизации функционально-структурных описаний цифровых устройств на основе продукционно-фреймовой модели представления знаний / П. Н. Бибило, В. И. Романов // Проблемы разработки перспективных микро- и наноэлектронных систем. – 2020. – Вып. 4. – С. 9–16.

[11] Бибило, П.Н. Экспериментальное сравнение эффективности программ минимизации систем булевых функций в классе дизъюнктивных нормальных форм / П.Н. Бибило, И.П. Логинова // Информатика. – 2022. – № 2(19). – С. 26–55.

[12] Logic Minimization Algorithm for VLSI Synthesis / K. R. Brayton [et al.]. – Boston : Kluwer Academic Publishers, 1984. – 193 p.

[13] Mishchenko, A. An Introduction to Zero-Suppressed Binary Decision Diagrams / A. Mishchenko. – Berkeley : University of California, 2014. – 15 p.

[14] Бибило, П.Н. Системы проектирования интегральных схем на основе языка VHDL. StateCAD, ModelSim, LeonardoSpectrum / П.Н. Бибило. – М.: СОЛОН-Пресс, 2005. – 384 с.

[15] Черемисинова, Л.Д. Программные средства верификации описаний комбинационных устройств в процессе логического проектирования / Л.Д. Черемисинова, Д.Я. Новиков // Программная инженерия, 2013, № 7, с. 8–15.

[16] Логинова, И.П. Верификация с использованием средств formalpro в системе логического проектирования заказных КМОП СБИС / И. П. Логинова // Новые информационные технологии в исследовании сложных структур : материалы Двенадцатой конференции с международным участием. 4–8 июня 2018 г. – Томск : Издательский Дом Томского государственного университета, 2018. – С. 72–73.

[17] Торопов, Н. Р. Минимизация систем булевых функций в классе ДНФ / Н. Р. Торопов // Логическое проектирование : сб. науч. тр. – Минск : Ин-т техн. кибернетики НАН Беларуси, 1999. – Вып. 4. – С. 4–19.

[18] Поттосин, Ю. В. Метод минимизации системы полностью определенных булевых функций / Ю. В. Поттосин, Н. Р. Торопов, Е. А. Шестаков // Информатика. – 2008. – № 2(18). – С. 102–110.

[19] Торопов, Н. Р. Преобразование многоярусной комбинационной сети в двухъярусную / Н. Р. Торопов // Логическое проектирование : сб. науч. тр. – Минск : Ин-т техн. кибернетики НАН Беларуси, 2000. – Вып. 5. – С. 4–14.

[20] E Ayguadé, E. . The design of Open MP tasks / E. Ayguade [et al.] // IEEE Transactions on Parallel and Distributed Systems. – 2009. – Vol. 20, no. 3. – P. 404 –418.

[21] Шлее, М. Qt 5.10. Профессиональное программирование на C++ / М. Шлее. – СПб.: БХВ-Петербург, 018–1072 с.

PERFORMANCE ANALYSIS FOR INPLEMENTING PARALLEL COMPUTING TO SOLVE OPTIMIZATION PROBLEMS IN A LOGICAL DESIGN SYSTEM

I.P. Loginova
Senior researcher, UIIP NASB, PhD, associate professor

United Institute of Informatics Problems of the National Academy of Sciences of Belarus (UIIP NASB), Belarus
E-mail: irilog@mail.ru

Abstract. The paper considers the possibility of implementing parallel computing in a system that performs for technologically independent optimization of functional and structural descriptions of logic circuits. Using the example of some logical optimization programs, the methodology for organizing parallel work is demonstrated, including those programs that are represented by executable code. A comparison of the performance of various options for running parallel optimization programs on a stream of known examples is carried out. The results of computational experiments on a multicore system with shared memory are presented. The effectiveness of the approach used in the organization of parallel computing in logical design systems for solving time-consuming tasks is confirmed.

Keywords: multicore system, parallel process, task scheduler, speedup of performance, Disjunctive Normal Form, logical optimization, VLSI.