УДК 004.272.2 (075.8)

*О. Н. КАРАСИК, А. А. ПРИХОЖИЙ*

# УСОВЕРШЕНСТВОВАННЫЙ ПЛАНИРОВЩИК КООПЕРАТИВНОГО ВЫПОЛНЕНИЯ ПОТОКОВ НА МНОГОЯДЕРНОЙ СИСТЕМЕ

*Белорусский национальный технический университет*

*Рассматриваются три архитектуры планировщика кооперативного выполнения потоков в многопоточном приложении, исполняемом на многоядерной системе. Архитектура A0 использует средства взаимодействия и синхронизации потоков, предоставляемые операционной системой. Архитектура A1 вводит новый примитив синхронизации потоков и единую для планировщика очередь заблокированных потоков, благодаря которым уменьшает активность взаимодействия потоков с операционной системой и значительно ускоряет процессы блокировки и разблокировки потоков. Архитектура A2 заменяет единую очередь заблокированных потоков на отдельные очереди для каждого примитива синхронизации и расширяет набор внутренних состояний примитива, уменьшая взаимозависимость потоков планирования и значительно ускоряя процессы блокировки и разблокировки рабочих потоков. Архитектуры планировщика реализованы в операционных системах Windows на базе технологии User Mode Scheduling. Важные экспериментальные результаты получены для многопоточных приложений, реализующих два блочно-параллельных алгоритма решения систем линейных алгебраических уравнений методом Гаусса. Алгоритмы различаются способами распределения данных между потоками и моделями синхронизации потоков. Число потоков варьировалось от 32 до 7936. Архитектура A1 показала ускорение до 8.65%, а архитектура A2 показала ускорение до 11.98 % по сравнению архитектурой A0 на блочно-параллельных алгоритмах с учетом их прямого и обратного хода. На обратном ходе алгоритмов архитектура A1 дала ускорение до 125 %, а архитектура A2 дала ускорение до 413 % по сравнению архитектурой A0. Эксперименты убедительно доказывают, что предлагаемые в статье архитектуры A1 и A2 выигрывают у A0 тем значительнее, чем большее количество блокировок и разблокировок потоков происходит во время выполнения многопоточного приложения.*

***Ключевые слова:*** *Многопоточное приложение, планировщик, кооперативная модель, многоядерная система*

## Введение

В современном мире, когда многоядерные системы распространены повсеместно, разработка эффективных многопоточных приложений является актуальной и востребованной задачей. Однако создание многопоточного приложения, способного эффективно использовать весь потенциал многоядерной системы, является трудоемкой задачей, требующей привлечения высококвалифицированных специалистов. С одной стороны, разработка эффективного многопоточного приложения, способного адаптироваться к возможностям конкретной аппаратной архитектуры, требует глубокого понимания механизмов работы операционной системы и всех аппаратных компонентов. С другой стороны, алгоритмы планирования потоков, реализуемые операционными системами, в большинстве своем достаточно универсальны и направлены на обеспечение производительности системы в целом, а не на достижение максимальной эффективности выполнения конкретного приложения. Зачастую они не учитывают особенности того или иного аппаратного компонента.

Поэтому для обеспечения переносимости и масштабируемости многопоточного приложения широкое распространение получили различные целевые библиотеки и платформы, реализующие алгоритмы планирования для эффективного выполнения многопоточных приложений с учетом режима многозадачности и в привязке к конкретной аппаратной архитектуре [1, 2, 3, 4].

В данной статье исследуются возможности повышения производительности библиотеки, предоставляющей средства разработки многопоточных приложений под операционными

системами (ОС) семейства Windows и поддерживающей кооперативную модель выполнения потоков. Эффективность разрабатываемых средств иллюстрируется на задаче решении систем линейных алгебраических уравнений (СЛАУ) блочно-параллельными методами Гаусса [5-6] и алгоритмами кооперативного оптимального управления выполнением взаимодействующих потоков [8].

### Базовая архитектура планировщика

В основу разрабатываемого планировщика пользовательских потоков положена технология User Mode Scheduling (UMS) [8], появившаяся в семействе операционных систем Windows, начиная с седьмой версии. Базовая архитектура А0 планировщика состоит из нескольких компонентов.

Менеджер памяти обеспечивает работу с оперативной памятью и наделен способностью точечного управления памятью планировщика при работе с NUMA (Non-Universal Memory Access) [9, 10], буферизации и т. д., а также способностью предотвращения взаимных блокировок между пользовательскими потоками и потоками планировщика.

Пользовательский поток (ПП) представляет собой надстройку над UmsThread и содержит информацию о конфигурации, состоянии и различных атрибутах, оказывающих влияние на обработку ПП планировщиком.

Поток планировщика (ППЛ) представляет собой надстройку над потоком операционной системы, выполняющимся в режиме UmsSchedulerThread. ППЛ конфигурирует выполнение ПП на логическом процессоре (ЛП), при этом, с целью группирования потоков, он использует очередь ОГПП готовых к выполнению ПП, очередь ОЗПП заблокированных ПП, а также указатель на единую очередь UmsCompletionList, предоставляемую ОС для передачи информации о соответствующих ПП UmsThread, отстранённых от выполнения по инициативе операционной системы (в случае блокировки по системным вызовам, ожидания выделения памяти, окончании выполнения и т.д.). Каждый логический процессор обслуживает свой ППЛ, дающий возможность выполнения любого потока ПП на данном логическом процессоре.

ОС Windows предоставляет широкий спектр примитивов синхронизации (мьютекс, событие, семафор и т.д.), однако блокировка UmsThread и соответствующего ПП осуществляется одинаково вне зависимости от типа примитива синхронизации.

Проанализируем процессы блокировки и разблокировки ПП (рис. 1) с использованием примитива синхронизации ПСОС, построенного средствами ОС. Процесс блокировки (рис. 1, *а*) осуществляется по цепочке ПП → ПСОС → ОС, при этом ПП вызывает одну из *wait* функций ОС и передает в нее дескриптор ПСОС. Вызов *wait* функции является системным, поэтому соответствующий ПП UmsThread блокируется ОС до его завершения. Вся цепочка выполняется без какого-либо взаимодействия ПП с ППЛ и реализуется посредством взаимодействия ПП с ОС.

В свою очередь, процесс разблокировки ПП с использованием ПСОС является достаточно сложным и осуществляется, с одной стороны, по цепочке ПП1 → ПСОС → ОС, посредством взаимодействия ПП1 с ПСОС и вызова функции, соответствующей данному примитиву синхронизации (рис. 1, *б*). ПСОС подает сигнал ОС на освобождение заблокированного потока UmsThread, получив который ОС, по цепочке ОС → UMSCL, добавляет разблокированный UmsThread в очередь UMSCL после чего, по цепочке ОС → ППЛ, посылает уведомление ППЛ (рис. 1, *б*). Получив уведомление, ППЛ по цепочке ППЛ → UMSCL → ОГПП → ПП, извлекает UmsThread из очереди UMSCL и сопоставляет разблокированный UmsThread с ПП. Далее ППЛ переводит разблокированный ПП в состояние «готов» и добавляет его в очередь ОГПП. После этого ППЛ

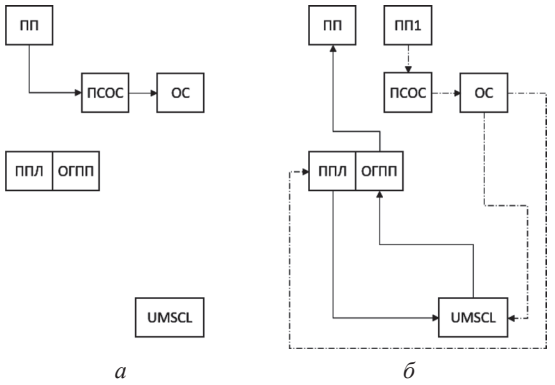



Рис. 1. Схематическое изображение процессов блокировки (*а*) и разблокировки (*б*) потока ПП с использованием базовой архитектуры А0 и примитива синхронизации ПСОС

извлекает ПП из ОГПП и переходит к его выполнению (рис. 1, *б*).

### Модифицированная архитектура планировщика

С целью упрощения и ускорения процессов блокировки и разблокировки потоков ПП по сравнению с базовыми процессами блокировки и разблокировки UmsThread в ОС, разработан новый примитив синхронизации (ПС) и новая архитектура A1 планировщика. Они учитывают возможности, предоставляемые UMS, и расширяют их до библиотеки «Планировщик», предоставляющей эффективные средства кооперативного управления выполнением пользовательских потоков. Отличительной особенностью новой архитектуры и ПС является исключение взаимодействия с ОС при управлении ПП.

Взаимодействие ПС с ПП происходит согласно алгоритму, показанному на рис. 2. Процесс блокировки осуществляется следующим образом (рис. 2, *а*). ПП проверяет текущее состояние ПС. Если ПС находится в состоянии «установлен», то ПП, по цепочке ПП → ПС → ПП, изменяет его состояние на «не установлен» и продолжает выполнение. Если ПС находится в состоянии «не установлен», то ПП, по цепочке ПП → ПС → ППЛ → ОЗПП, регистрирует исполняющий его ППЛ для получения уведомления от данного ПС при переводе его с состояние «установлен» и возвращает управление исполняющему ППЛ, который изменяет состояние ПП на «заблокирован» и добавляет его в ОЗПП (рис. 2, *а*).

Процесс разблокировки происходит следующим образом (рис. 2, *б*). Поток ПП1, по цепочке ПП1 → ПС → ППЛ, переводит ПС в состояние «установлен» и посылает каждому зарегистрированному ППЛ уведомление о том, что данный ПС находится теперь в этом состоянии. ППЛ, получив уведомление от ПС, проверяет каждый ПП, находящийся в ОЗПП, на возможность его разблокировки. Все разблокированные ПП переводятся, по цепочке ППЛ → ОЗПП → ОГПП, в очередь ОГПП и становятся доступными для дальнейшего выполнения (рис. 2, *б*). Затем ППЛ выбирает из ОГПП следующий ПП и переходит к его непосредственному исполнению.

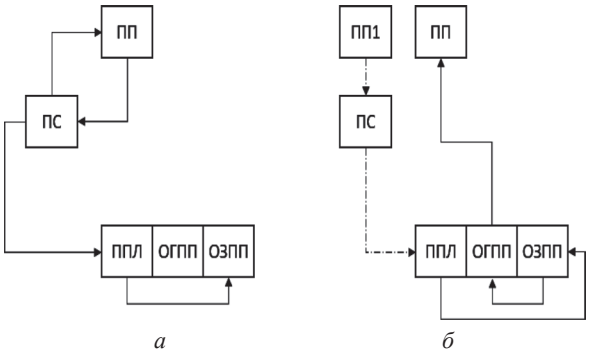



Рис. 2. Схематическое изображение процессов блокировки (*а*) и разблокировки (*б*) потока ПП с использованием модифицированной архитектуры A1 и примитива синхронизации ПС

### Анализ модифицированной архитектуры

С целью выявления достоинств и недостатков модифицированной архитектуры A1 планировщика потоков были проведены вычислительные эксперименты с многопоточными приложениями, решающими прикладные задачи. В частности, эксперименты над двумя блочно-параллельными алгоритмами Метод1 и Метод2 [5–7], реализующими метод Гаусса решения СЛАУ, показали, что время выполнение обратного хода как для Метода 1, так и для Метода 2 сократилось для каждого количества потоков (рис. 3).

В то же время, анализ модифицированной архитектуры планировщика A1 выявил следующие факторы, оказывающие негативное влияющие на его производительность:

1. Для реализации процессов блокировки и разблокировки ПП с использованием ПС необходима регистрация каждого ППЛ с целью получения уведомления о переводе ПС в состояние «установлен».

2. Разблокировка каждого ПП, получившего уведомление от ПС, требует поиска, осуществляемого ППЛ в очереди ОЗПП.

3. Обеспечение корректного доступа к единой очереди UMSCL и возможность одновременного доступа нескольких ППЛ к одному ПП сериализуют процессы планирования, что, в свою очередь, приводит к замедлению работы многопоточного приложения.

Выявленные в процессе анализа факторы снижения производительности многопоточного приложения обосновывают необходимость дальнейшего совершенствования планировщика.

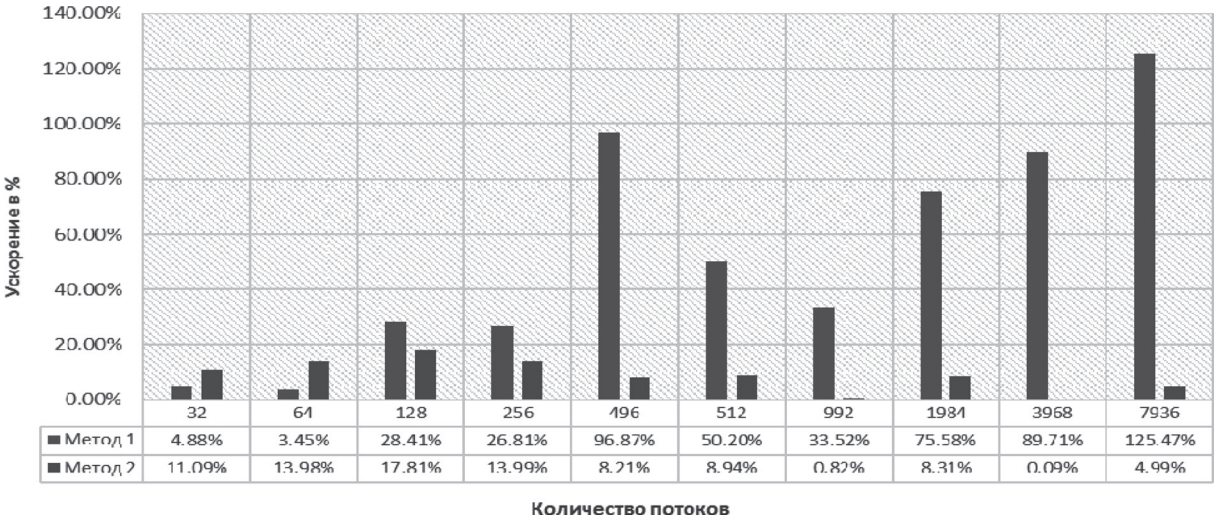



Рис. 3. Ускорение в процентах выполнения обратного хода в алгоритмах Метод1 и Метод2 решения СЛАУ усовершенствованным планировщиком А1 по сравнению с планировщиком А0 в зависимости от количества пользовательских потоков

## Усовершенствованная архитектура планировщика

С целью повышения производительности планировщика, архитектура А1 усовершенствована и доработана до архитектуры А2 (рис. 4). Очередь UMSCL, единая для всех ППЛ, использовалась в архитектуре А1 для более быстрой реакции планировщика на добавление UmsThread в очередь UMSCL. Однако при этом возникает проблема корректного доступа к единой очереди UMSCL большого размера, порождающая проблему сериализации работы планировщика при обеспече нии доступа к ПП. ППЛ использует механизм сериализации, реализованный посредством атомарных операций CAS. В архитектуре А2 проблема сериализации решается созданием отдельной очереди UMSCL для каждого ППЛ, обслуживающего отдельный логический процессор, и в перемещении очереди заблокированных потоков ОЗПП из потока планирования ППЛ в примитив синхронизации ПС. Такая модификация архитектуры исключает полный обход всей очереди ОЗПП при получении уведомления от ПС и упрощает задачу поиска ПП для разблокировки. Она значительно сокращает время разблокировки и объем работы, выполняемый ППЛ, и не требует регистрации ППЛ в ПС.

Для ускорения процесса блокировки ПП, в примитив синхронизации ПС введены дополнительные состояния. Среди них два конечных состояния «установлен» и «не установлен», и два переходных состояния «устанавливается» и «присоединяется». «Устанавливается» означает переход ПС из состояния «не установлен» в состояние «установлен» и освобождение ПП из ОЗПП данного ПС. «Присоединяется» означает то, что ППЛ осуществляет процесс добавления ПП в ОЗПП данного ПС. Состояния «устанавливается» и «присоединяется» имеют по два подсостояния, описывающих ситуации, когда в процессе установки или присоединения ПП другой ПП1 пытается перевести ПС в состояние «установлен» или «не установлен». Поэтому ПП1, пытающийся изменить состояние ПС должен повторить запрос по завершении процесса ПП. Такой набор состояний позволяет нескольким ПП одновременно обращаться к одному ПС, и позволяет нескольким ППЛ выполнять процедуру добавления ПП в ОЗПП данного ПС без блокировки и без сериализации доступа к ПС.

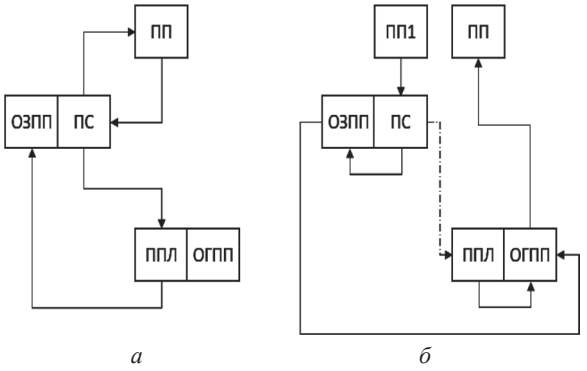
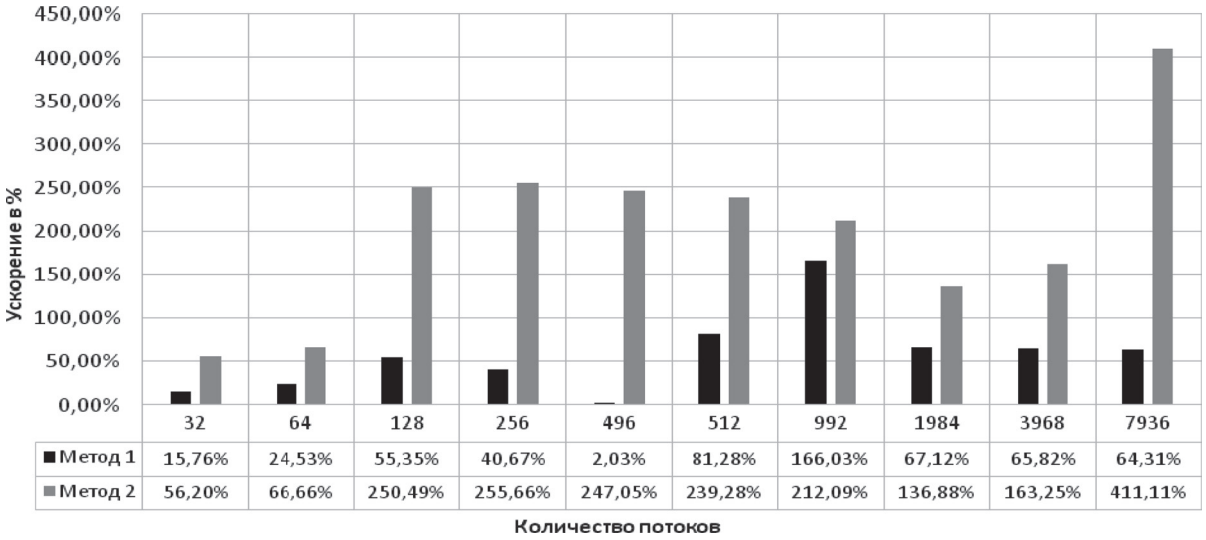



Рис. 4. Схематическое изображение процессов блокировки (*а*) и разблокировки (*б*) потока ПП с использованием усовершенствованной архитектуры А2




| | 32 | 64 | 128 | 256 | 496 | 512 | 992 | 1984 | 3968 | 7936 |
|---|---|---|---|---|---|---|---|---|---|---|
| Метод 1 | 15,76% | 24,53% | 55,35% | 40,67% | 2,03% | 81,28% | 166,03% | 67,12% | 65,82% | 64,31% |
| Метод 2 | 56,20% | 66,66% | 250,49% | 255,66% | 247,05% | 239,28% | 212,09% | 136,88% | 163,25% | 411,11% |

Рис. 5. Ускорение в процентах выполнения обратного хода в алгоритмах Метод1 и Метод2 решения СЛАУ усовершенствованным планировщиком А2 по сравнению с планировщиком А1 в зависимости от количества пользовательских потоков

С целью ускорения процессов блокировки и разблокировки потоков, алгоритмы взаимодействия ПП и ПС полностью пересмотрены (рис. 4). Теперь заблокированные ПП хранятся в отдельной для каждого ПС очереди ОЗПП вне потока планирования ППЛ. Процесс блокировки показан на рис.4а. ПП проверяет текущее состояние ПС. Если ПС находится в состоянии «не установлен», то ПП возвращает управление исполняющему ППЛ с указанием осуществления его блокировки на данном ПС. По цепочке ПП → ПС → ППЛ → ОЗПП, поток планирования ППЛ переводит ПС в состояние «заблокирован» и добавляет его в очередь ОЗПП данного ПС. Процесс разблокировки показан на рис. 4, *б*. ПП1 переводит ПС в состояние «устанавливается». Если ОЗПП содержит хотя бы один ПП, то, по цепочке ПП1 → ПС → ОЗПП → ОГПП, ПП1 извлекает ПП из очереди ОЗПП, переводит в состояние «готов» и добавляет в ОГПП потока планирования ППЛ, обсуживающего логический процессор, на котором будет исполняться ПП. После этого примитив синхронизации ПС переводится в состояние «не установлен». Если ОЗПП не содержит ни одного потока ПП, то ПП1 переводит ПС в состояние «установлен».

### Экспериментальная среда

Эксперименты над планировщиком потоков выполнены на многоядерной системе, оснащенной двумя процессорами Intel®Xeon® CPUE5520 и оперативной памятью 16 GB, работающей с частотой 1 GHz. Каждый процессор включает 4 ядра, работающих с частотой 2.26 GHz и оснащенных технологией Hyper-Threading Technology. Каждый физический процессор имеет разделяемую между ядрами кэш память емкостью 8 MB, а каждое ядро имеет локальную кэш память первого уровня емкостью 64KB и второго уровня емкостью 256 KB. Кроме того, каждый из 2-х физических процессоров выполняет доступ к локальной и удаленной памяти по принципу точка-точка с использованием технологии NUMA. Благодаря технологии QPI (Quick Path Interconnect) каждый процессор обладает интегрированным контроллером для работы с памятью. Управление многоядерной системой осуществляется ОС Windows Server 2012 R2 64.

### Результаты вычислительных экспериментов

Рис. 3 сравнивает результаты работы модифицированного планировщика А1 с результатами работы базового планировщика А0, полученными при решении СЛАУ блочно-параллельными алгоритмами Метод1 и Метод2 [5–7], реализованными в виде многопоточных приложений. Рис. 5 дает сравнение усовершенствованного планировщика А2 с модифицированным планировщиком А1 на тех же алгоритмах решения СЛАУ. Совместный анализ рис. 3 и 5 позволяет выполнить сравнение А2 с А0. Заметим, что алгоритмы Метод1 и Метод2 различаются разбиением программного кода и эле-

ментов данных по потокам, количество которых варьируется от 32 до 7936.

При решении СЛАУ алгоритмом Метод1 с использованием планировщика A1 полученное ускорение выполнения обратного хода по сравнению с планировщиком A0 составило от 3.45% (64 потока) до 125.47% (7936 потоков). Для 3968 потоков при учете прямого и обратного хода процесса решения СЛАУ время выполнения сократилось с 404.99 сек до 375.83 сек, а ускорение составило 7.76%. Что касается обратного хода, его время сократилось с 27.57 сек до 14.54 сек, а ускорение составило 89.71%. В свою очередь ускорение, полученное планировщиком A2 по сравнению с планировщиком A1, варьировалось от 2.03% (496 потоков) до 166.03% (992 потока). Для 3968 потоков, при учете прямого и обратного хода процесса решения СЛАУ, время выполнения сократилось с 375.83 сек до 367.1 сек, а ускорение составило 2.38%. Чистое же время обратного хода сократилось с 14.54 сек до 8.77 сек, при этом получено ускорение 65.82%. Таким образом, для 3968 потоков общее время выполнения с использованием усовершенствованной архитектуры планировщика A2 по сравнению с базовой архитектурой A0 сократилось с 404.9 сек до 367.1 сек, а ускорение составило 10.33%. Чистое же время обратного хода сократилось с 27.57 сек до 8.77 сек, при этом получено ускорение 214.57%.

При решении СЛАУ алгоритмом Метод2 с использованием модифицированной архитектуры планировщика A1 полученное ускорение выполнения обратного хода по сравнению с базовой архитектурой планировщика A0 составило от 0.09% (3968 потока) до 17,81% (128 потоков). Для 3968 потоков, при учете прямого и обратного хода процесса решения СЛАУ, время выполнения сократилось с 413.14 сек до 380.26 сек, а ускорение составило 8.65%. Чистое же время обратного хода сократилось с 6.0695 сек до 6.0639 сек, при этом получено ускорение 0.09%. В свою очередь ускорение, полученное с использованием усовершенствованной архитектуры планировщика A2 по сравнению с модифицированной архитектурой планировщика A1 составило от 56.20% (32 потока) до 411.11% (7936 потоков). Для 3968 потоков, при учете прямого и обратного хода процесса решения СЛАУ, процессорное время сократилось с 380.26 сек до 368.92 сек, а ускорение составило 3.07%. Чистое же время обратного хода сократилось с 6.0639 сек до 2.3035 сек, при этом получено ускорение 163.25%. Таким образом, для 3968 потоков общее время выполнения с использованием усовершенствованной архитектуры планировщика A2 по сравнению с базовой архитектурой A0 сократилось с 413.14 сек до 368.92 сек, а ускорение составило 11.98%. Чистое же время обратного хода сократилось с 6.0695 сек до 2.3035 сек, при этом получено ускорение 163.49%.

Заметим, что прирост производительности получен для всех конфигураций потоков, использованных при проведении экспериментов.

## Заключение

Повышение производительности планировщика пользовательских потоков в системе с кооперативной многозадачностью достигнуто усовершенствованием его архитектуры и алгоритма работы. Изменение процессов блокировки и разблокировки потоков с использованием предлагаемого примитива синхронизации, а также ряд других изменений, затронувших основные компоненты планировщика, позволили значительно сократить время выполнения блочно-параллельных алгоритмов решения СЛАУ методом Гаусса и получить ускорение обратного хода до 413% при общем ускорении решения системы от 10.33% до 11.98%. Модернизация планировщика позволила получить не только прирост производительности на экспериментальной многоядерной системе, но также улучшить масштабируемость планировщика в расчете на более мощные многоядерные системы, где количество потоков может возрастать значительно.

## Литература

1. **Rajagopalan, M.** Thread Scheduling for Multi-Core Platforms / M. Rajagopalan, B. T. Lewis, and T. A. Anderson, // HOTOS'07 Proc. of the 11th USENIX workshop on Hot topics in operating systems, San Diego, CA, 2007.

2. **Yun, H.** Deterministic Real-time Thread Scheduling / H. Yun, C. Kim, and L. Sha // preprint arXiv: 1104.2110, 2011.

3. **Shelepov, D.** Scheduling on heterogeneous multicore processors using architectural signatures / D. Shelepov, and A. Fedorova // Proc. Workshop on the Interaction between Operating Systems and Computer Architecture, in conjunction with ISCA-35, Beijing, China, 2008.

4. **Fedorova, A.** Cache-fair thread scheduling for multicore processors / A. Fedorova, M. I. Seltzer, and M. D. Smith // Harvard Computer Science Group Technical Report TR-17-06, 2006.

5. **Прихожий, А. А.** Исследование методов реализации многопоточных приложений на многоядерных системах / А. А. Прихожий, О. Н. Карасик // Информатизация образования, 2014, № 1. – С. 43–62.

6. **Прихожий, А. А.** Кооперативная модель оптимизации выполнения потоков на многоядерной системе / А. А. Прихожий, О. Н. Карасик // Системный анализ и прикладная информатика, 2014, № 4. – С. 13–20.

7. **Прыхожы А. А.** Кааператыўныя блочна-паралельныя алгарытмы рашэння задач на шмат'ядравых сістэмах / А. А. Прыхожы, А. М. Карасік // Системный анализ и прикладная информатика, 2015, № 2. – С. 10–18.

8. **Al-Rayes, H. T.** Concurrent Programming in Windows Vista / H. T. Al-Rayes // International Journal of Electrical & Computer Sciences IJECS-IJENS. Vol. 12. No. 5, pp. 32–37.

9. **Manchanda, Nakul, and Karan Anand.** Non-uniform memory access // New York University, 2010. – 4 p.

10. **Rajput, V.** Performance Analysis of UMA and NUMA Models / V. Rajput, S. Kumar and V. K. Patle. // IJCSIT. Vol. 2, Issue 10, pp. 1457–1458.

## References

1. **Rajagopalan, M.** Thread Scheduling for Multi-Core Platforms / M. Rajagopalan, B. T. Lewis, and T. A. Anderson, // HOTOS'07 Proc. of the 11th USENIX workshop on Hot topics in operating systems, San Diego, CA, 2007.

2. **Yun, H.** Deterministic Real-time Thread Scheduling / H. Yun, C. Kim, and L. Sha // preprint arXiv: 1104.2110, 2011.

3. **Shelepov, D.** Scheduling on heterogeneous multicore processors using architectural signatures / D. Shelepov, and A. Fedorova // Proc. Workshop on the Interaction between Operating Systems and Computer Architecture, in conjunction with ISCA-35, Beijing, China, 2008.

4. **Fedorova, A.** Cache-fair thread scheduling for multicore processors / A. Fedorova, M. I. Seltzer, and M. D. Smith // Harvard Computer Science Group Technical Report TR-17-06, 2006.

5. **Prihozhy, A. A.** Investigation of implementation methods of multi-thread applications on multi-core systems / A. A. Prihozhy, O. N. Karasik // Informatization of Education, 2014, № 1. – С. 43–62.

6. **Prihozhy, A. A.** Cooperative model of optimal thread execution on multi-core system / A. A. Prihozhy, O. N. Karasik // System analysis and applied computer science, 2014, № 4. – С. 13–20.

7. **Prihozhy, A. A.** Cooperative block-parallel algorithms for solving tasks on multi-core systems / A. A. Prihozhy, O. N. Karasik // System analysis and applied computer science, 2015, № 2. – С. 10–18.

8. **Al-Rayes, H. T.** Concurrent Programming in Windows Vista / H.T. Al-Rayes // International Journal of Electrical & Computer Sciences IJECS-IJENS Vol.12 No.5, pp. 32–37.

9. **Manchanda, Nakul, and Karan Anand.** Non-uniform memory access // New York University, 2010. – 4 p.

10. **Rajput, V.** Performance Analysis of UMA and NUMA Models / V. Rajput, S. Kumar and V. K. Patle. // IJCSIT Vol. 2, Issue 10, pp. 1457–1458.

| *Поступила* | *После доработки* | *Принята к печати* |
|---|---|---|
| *26.12.2016* | *25.02.2017* | *06.03.2017* |

*Karasik O., Prihozhy A.*

# ADVANCED SCHEDULER FOR COOPERATIVE EXECUTION OF THREADS ON MULTI-CORE SYSTEM

*Belarusian National Technical University*

*Three architectures of the cooperative thread scheduler in a multithreaded application that is executed on a multi-core system are considered. Architecture A0 is based on the synchronization and scheduling facilities, which are provided by the operating system. Architecture A1 introduces a new synchronization primitive and a single queue of the blocked threads in the scheduler, which reduces the interaction activity between the threads and operating system, and significantly speed up the processes of blocking and unblocking the threads. Architecture A2 replaces the single queue of blocked threads with dedicated queues, one for each of the synchronizing primitives, extends the number of internal states of the primitive, reduces the interdependence of the scheduling threads, and further significantly speeds up the processes of blocking and unblocking the threads. All scheduler architectures are implemented on Windows operating systems and based on the User Mode Scheduling. Important experimental results are obtained for multithreaded applications that implement two blocked parallel algorithms of solving the linear algebraic equation systems by the Gaussian elimination. The algorithms differ in the way of the data distribution among threads and by the thread synchronization models. The number of threads varied from 32 to 7936. Architecture A1 shows the acceleration of up to 8.65% and the architecture A2 shows the acceleration of up to 11.98% compared to A0 architecture for the blocked parallel algorithms computing the triangular form and performing the back substitution. On the back substitution stage of the algorithms, architecture A1 gives the acceleration of up to 125%, and architecture A2 gives the acceleration of up to 413% compared to architecture A0. The experiments clearly show that the proposed architectures, A1 and A2 outperform A0 depending on the number of thread blocking and unblocking operations, which happen during the execution of multi-threaded applications. The conducted computational experiments demonstrate the improvement of parameters of multithreaded applications on a heterogeneous multi-core system due the proposed advanced versions of the thread scheduler.*

***Keywords:*** *Multi-threaded application, scheduler, cooperative model, multi-core system.*

**Карасик Олег Николаевич** – аспирант кафедры «Программное обеспечение вычислительной техники и автоматизированных систем» БНТУ, ведущий инженер программист компании «EPAM Systems», научные интересы в области параллельных многопоточных приложений и распараллеливания для многоядерных и многопроцессорных систем.

**Karasik Aleh** is a postgraduate of the Computer and system software department of Belarusian national technical university, and a leading software engineer at EPAM Systems. His research interests include parallel multithreaded applications and the parallelization for multicore and multiprocessor systems**.**

**Прихожий Анатолий Алексеевич** – профессор кафедры программного обеспечения вычислительной техники и автоматизированных систем БНТУ, доктор технических наук (1999), профессор (2001). Научные интересы в области языков программирования и описания цифровой аппратауры, распараллелтвающих компиляторов, инструментальных средств проектирования программных и аппаратных систем на логическом, поведенческом и системном уровнях. Имеет более 300 научных публикаций в Восточной и Западной Европе, США и Канаде.

**Anatoly Prihozhy** is a full professor at the Computer and system software department of Belarusian national technical university, doctor of science (1999) and professor (2001). His research interests include programming and hardware description languages, parallelizing compilers, and computer aided design tools for software and hardware at logic, high and system levels. He has over 300 publications in Eastern and Western Europe, USA and Canada.