Автономная некоммерческая организация высшего образования
«Университет Иннополис»

АННОТАЦИЯ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
(БАКАЛАВРСКУЮ РАБОТУ)
ПО НАПРАВЛЕНИЮ ПОДГОТОВКИ
09.03.01 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

НАПРАВЛЕННОСТЬ (ПРОФИЛЬ) ОБРАЗОВАТЕЛЬНОЙ ПРОГРАММЫ
«ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА»

Тема **Оптимизированный гетерогенный планировщик, управляемый графами переходов состояний**

Выполнил **Муталапов Арсен Ильдарович**

подпись

Иннополис, Innopolis, 2025

Оглавление

Аннотация

Все более широкое использование гетерогенных мобильных платформ заставляет разрабатывать более эффективные планировщики. Такие платформы состоят из вычислительных устройств, различающихся по времени работы и энергопотреблению. Задача планировщика - найти подходящий компромисс между временем выполнения и энергопотреблением. Мы предлагаем новые оптимальные и субоптимальные методы составления расписания, основанные на понятии графов переходов состояний, которые моделируют возможные расписания выполнения программ на различных типах вычислительных устройств. В качестве цели планирования мы выбрали метрику Energy-Delay Product, которая объединяет время и энергопотребление. Наши планировщики реализованы с помощью инструмента C++ Scheduler, который доступен по запросу.

Глава 1

Введение

В классическом однородном многоядерном планировании планировщик операционной системы распределяет доступные однородные ядра между динамически поступающими программами [1]. Когда количество ожидающих выполнения программ превышает количество ядер, планировщик распределяет программы по приоритетам, либо по статическим приоритетам, установленным разработчиком системы, либо по конкретным динамически вычисляемым приоритетам. Целями планировщика являются минимизация среднего времени выполнения программ, задержки, числа миграций и вытеснений и другие. Мы подчеркиваем, что проблема оптимального планирования работы многоядерных систем является NP-трудной, и поэтому эвристические подходы обычно используются для поиска субоптимального решения.

1.1 Гетерогенное планирование

Кроме того, в отличие от классического гомогенного случая, гетерогенное планирование значительно усложняется наличием нескольких типов

вычислительных блоков для гетерогенной аппаратной платформы [2]. Некоторые из этих гетерогенных блоков медленные, но энергоэффективные, а другие быстрые, но слишком энергозатратные (см. рис. 1.1). Известным примером гетерогенной платформы является архитектура ARM big.LITTLE [3], концептуально изображенная на рис. 1.2a. Она состоит из:

- кластер быстрых (“больших”), но энергоемких ядер;
- кластер медленных (“маленьких”), но энергоэффективных ядер;
- GPU и
- устройства NPU.

Посмотрите на сравнение производительности “больших” (Cortex-A15) и “маленьких” (Cortex-A7) ядер на рис. 1.2b. Для этих типов ядер существует пересекающийся диапазон производительности, в котором использование “маленьких” ядер приводит к гораздо меньшему энергопотреблению, чем использование “больших” ядер. Соответствующее распределение этих вычислительных блоков зависит от активности программ. Например, чтение 32 байт из 8-Кбайт SRAM потребляет в 125 раз меньше энергии, чем из DRAM [4] Таким образом, гетерогенный планировщик направлен не только на оптимизацию временных аспектов, но и на оптимизацию общего энергопотребления. Эта проблема оптимизации гетерогенного планирования особенно актуальна для систем с автономными источниками питания.

Рисунки, как и остальная часть работы, опираются на графические обозначения, приведенные на рис. 1.3.

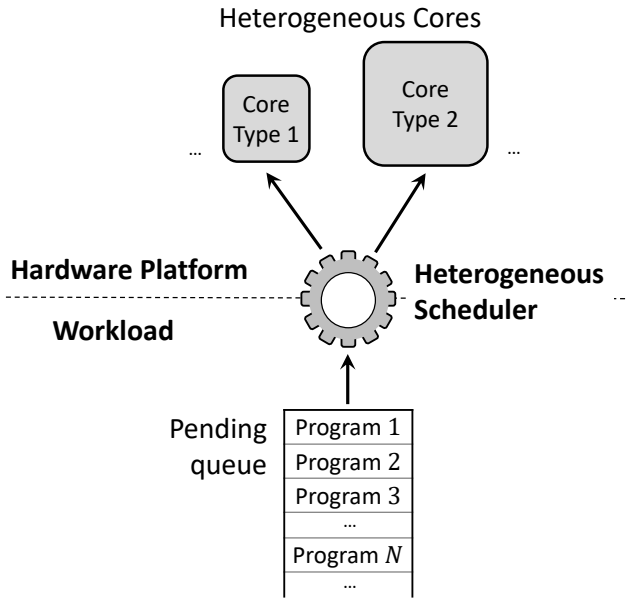
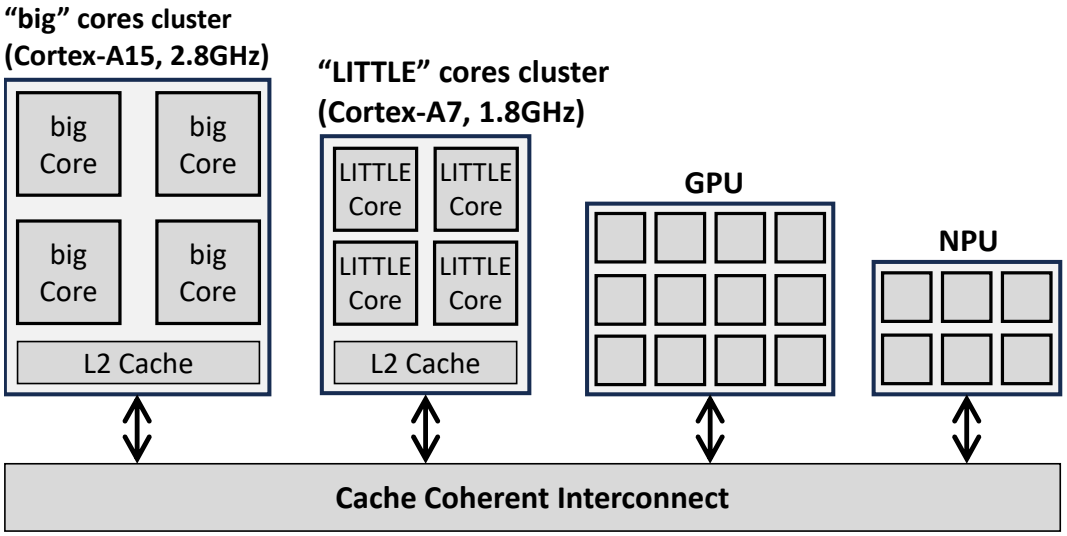


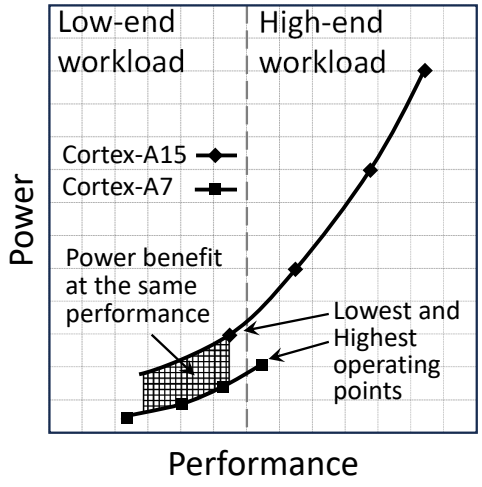
Рис. 1.1: Ключевые компоненты гетерогенного планирования

Программы Блоки		Время, сек		Энергия, Джоули	
		Ядро 1 (медленное)	Ядро 2 (быстрое)	Ядро 1	Ядро 2
P1	B1	4	3	8	9
	B2	5	3	3	4
P2	B1	5	4	6	10
	B2	3	2	3	7
	B3	3	2	3	6
P3	B1	3	2	6	7

Таблица 1.1: Требования исполнения программ по их блокам



(a) ARM big.LITTLE архитектура: обзор



(b) Тренды в производительности и энергопотреблении

Рис. 1.2: Гетерогенная архитектура: пример Cortex-A15 и Cortex-A7

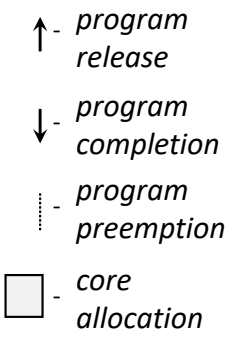
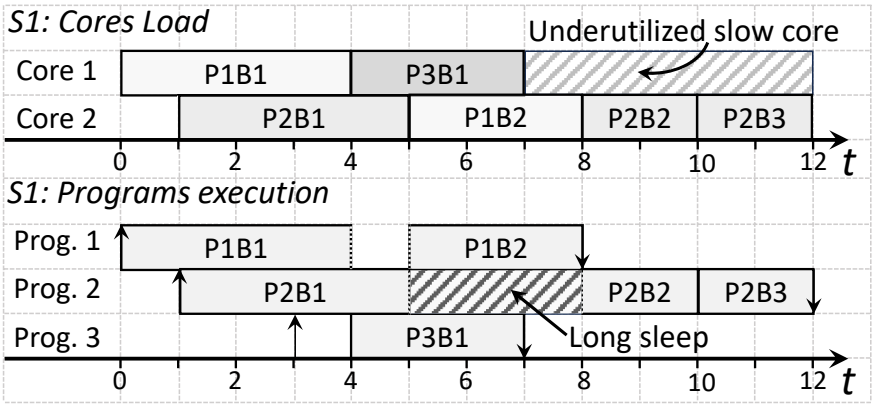


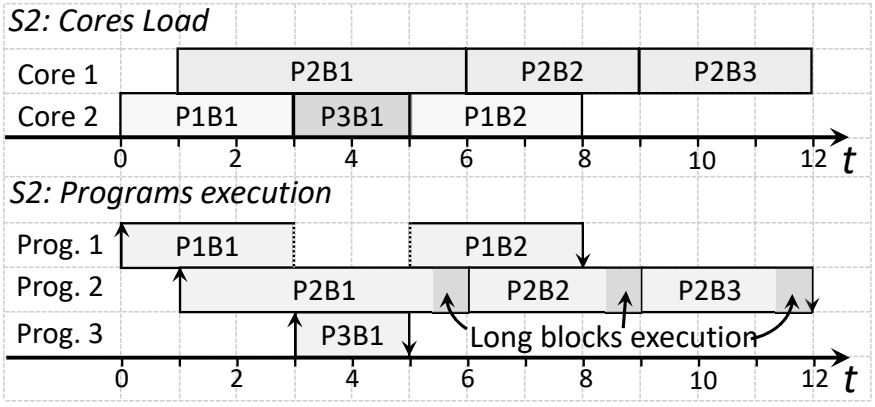
Рис. 1.3: Графическая нотация



(a) Schedule S1: исполнение программ

Schedule	Program	Latency	Execution Time	Response Time	Average Execution Time	Average Response Time
S1	P1	0	7	8	**6**	**7.67**
	P2	0	8	11		
	P3	1	3	4		

(b) S1: временные метрики



(c) Schedule S2: исполнение программ

Schedule	Program	Latency	Execution Time	Response Time	Average Execution Time	Average Response Time
S2	P1	0	6	8	**6.33**	**7**
	P2	0	11	11		
	P3	0	2	2		

(d) S2: временные метрики

Рис. 1.4: Примеры планирования программ для требований из таблицы 1.1 (миграции и преемственность не учтены)

Глава 2

Методология

Далее приведем игрушечный пример планирования трех программ на гетерогенной аппаратной платформе с одним “медленным” и одним “быстрым” ядром. Мы предполагаем, что программы состоят из отдельных логически разделенных блоков кода[1], а требования к их выполнению перечислены в таблице 1.1. На рис. 1.4a и 1.4c мы изобразили загрузку ядер и выполнение программ для двух возможных расписаний S1 и S2, предполагая, что программы поступают асинхронно в моменты времени 0, 1 и 3 соответственно. Например, в S1 “медленное” ядро 1 выделено для выполнения программ 1 и 3, а в S2 это ядро выполняет только все блоки программы 2.

[1]Понятие блока еще не определено и будет уточнено позже. Пока мы предполагаем, что блоки программы выполняются последовательно

Глава 3

Вклад

Мы предлагаем новый подход к гетерогенному планированию. Он основан на обходе графов переходов состояний выполняемых программ, который изображён на рис. 3.1. Более конкретно, наш вклад заключается в следующем:

- Теоретически оптимальный гетерогенный планировщик. В отличие от других существующих планировщиков, наш учитывает не только время отклика программ, но и потребление энергии;
- Эвристика планирования, которая позволяет принимать решения о планировании значительно быстрее, чем оптимальный планировщик. Мы также демонстрируем, что некоторые эвристики близки к оптимальным;
- Прототип инструмента на C++, реализующего все наши планировщики;
- Подробное сравнение всех предложенных планировщиков.

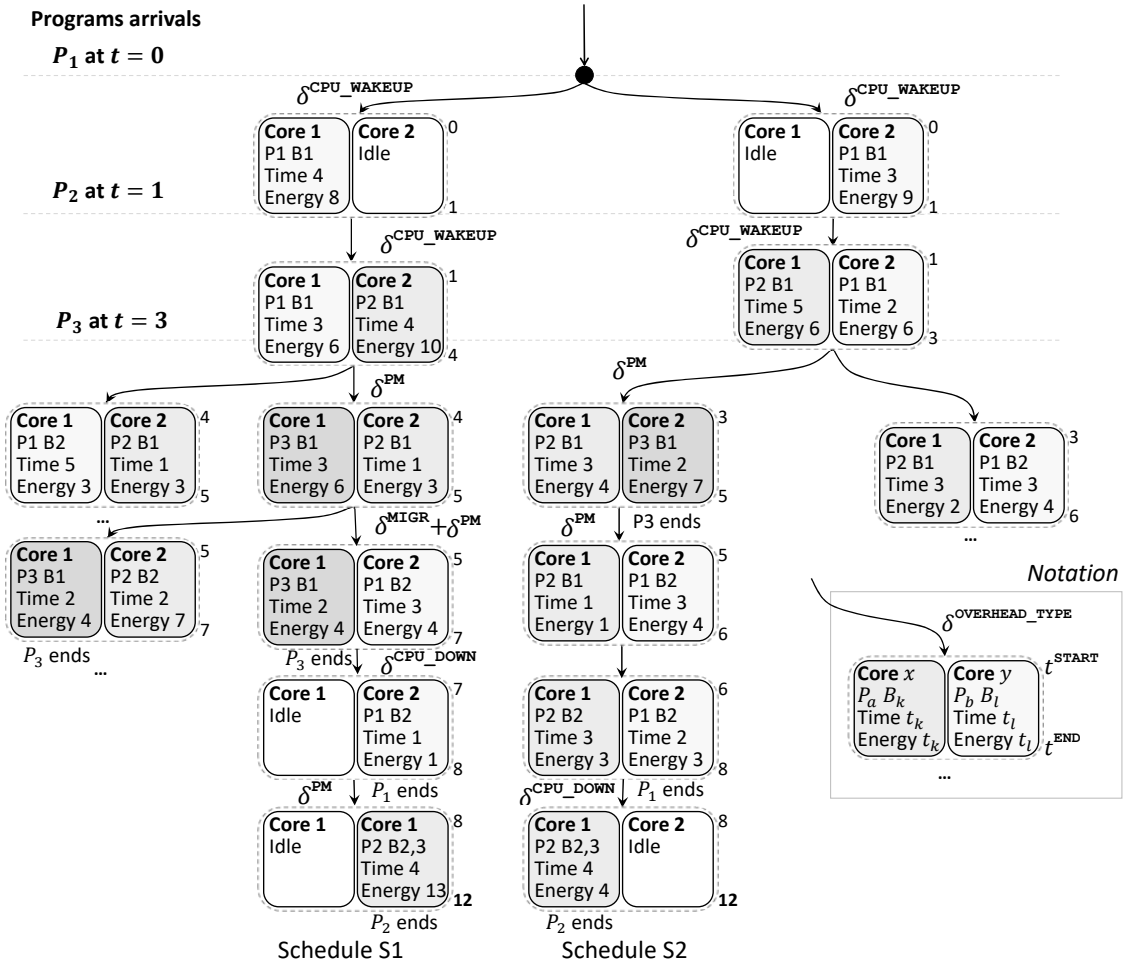


Рис. 3.1: Объединённый граф программ с расписаниями из рис. 1.4. Случай безпреемственного планирования

3.1 Ограничения

На данный момент наши планировщики не учитывают накладные расходы на преемственность и миграцию. Однако предложенная методология графов переходов состояний легко расширяется до таких более реалистичных сценариев.

Глава 4

Выводы

Мы решили проблему эффективного планирования работы гетерогенных систем с учетом времени отклика программ и энергопотребления. Мы предложили несколько планировщиков, среди которых есть оптимальный и эвристический: случайный и жадный. Наш ключевой вклад - учет энергопотребления, в отличие от других существующих планировщиков, таких как HEFT и HASS.

4.1 Детальный обзор вклада

В основе наших планировщиков лежит обход графа переходов состояний программ. Оптимальный планировщик должен обойти весь огромный по размеру граф, что требует значительного времени вычислений. Эвристики обходят только выбранные фрагменты графа, стремясь сократить время вычислений, но создают менее эффективные расписания. Чтобы проверить эффективность планировщиков, мы реализовали инструмент C++ Планировщик для моделирования параллельного выполнения входных программ. Так-

же мы реализовали инструмент для генерации параметров входных программ для проведения экспериментов. Эффективность планировщиков оценивается в ходе серии экспериментов в терминах произведения энергии и задержки (EDP), которое учитывает как время отклика программ, так и потребляемую энергию. Мы показали, что эвристика 100-случайных прогулок близка к оптимальному расписанию, в то время как жадная эвристика показывает наихудшую эффективность. Что касается времени выполнения, то вычисление оптимального расписания занимает на порядки больше времени из-за экспоненциальной сложности по сравнению с эвристиками. Отклонение EDP для расписаний составляет 40-50% для выбранных экспериментальных настроек. Мы пока не учитываем накладные расходы на миграцию и преемственность, которые должны быть включены.

Инструменты C++, используемые в данной работе доступно по требованию.

Список использованной литературы

[1] J. Anderson, J. Calandrino и U. Devi, «Real-Time Scheduling on Multicore Platforms,» в *12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06)*, 2006, с. 179—190. DOI: 10.1109/RTAS.2006.35.

[2] A. Radulescu и A. van Gemund, «Fast and effective task scheduling in heterogeneous systems,» в *Proceedings 9th Heterogeneous Computing Workshop (HCW 2000) (Cat. No.PR00556)*, 2000, с. 229—238. DOI: 10.1109/HCW.2000.843747.

[3] E. L. Padoin, L. L. Pilla, M. Castro, F. Z. Boito, P. O. Alexandre Navaux и J.-F. Méhaut, «Performance/energy trade-off in scientific computing: the case of ARM big.LITTLE and Intel Sandy Bridge,» *IET Computers & Digital Techniques*, т. 9, № 1, с. 27—35, 2015. DOI: https://doi.org/10.1049/iet-cdt.2014.0074. eprint: https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/iet-cdt.2014.0074. url: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-cdt.2014.0074.

[4] J. L. Hennessy и D. A. Patterson, *Computer Architecture, Sixth Edition: A Quantitative Approach*, 6th. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2017, ISBN: 0128119055.