4. МНЕНИЯ

4.1. В АРХИТЕКТУРЕ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ МЫ БЫЛИ И ОСТАЕМСЯ ПЕРВЫМИ

Бабаян Б.А.
Член-корреспондент РАН, директор по архитектуре подразделения
Core and Visual Computing Group корпорации Intel,
Intel Fellow.

Настоящая статья сформирована из высказываний Бориса Арташесовича Бабаяна и одобрена им, а потому подается как его авторский текст. Большая часть вошедших в статью высказываний взята из аудиозаписи разговора между ним и группой сотрудников журнала (А.Н. Козырев, И.В. Неволин, Н.В. Ноакк) 23.09.2018. Реплики других участников разговора вставлены как врезки в текст статьи. Также врезками даны справки о людях и фактах, упоминаемых в данном разговоре без пояснений. Готовясь к этому разговору, мы составили список вопросов, которые предполагали задать выдающимся разработчикам отечественной вычислительной техники, а также изучили более ранние интервью и высказывания Б.А. Бабаяна, связанные с историей факультета радиотехники и кибернетики МФТИ. Отсюда возникла идея – на основе всего сказанного подготовить статью для журнала «Цифровая экономика».

Первый студент МФТИ

Иногда хвалюсь тем, что я, наверно, – первый студент Физтеха, поскольку поступил на Физтех в 1951 году. Более того, я думаю, что этот факультет Физтеха был первым факультетом по вычислительной технике в мире. Сначала я был студентом ФТФ МГУ, а потом этот факультет выделился из МГУ и стал самостоятельным вузом. Еще я все время вспоминаю работу с Сергеем Алексеевичем Лебедевым.

Студент Б.А. Бабаян(слева) в студенческом лагере. 1973г.

В 1952 году я начал проходить учебную практику в ИТМиВТ АН СССР. Мы с коллективом работали в ИТМиВТ до начала 90-х, пока границы не открылись, и немного после[1]. В Intel мы перешли уже в 2004 году. И работа над вычислительными комплексами (Эльбрусами) – это великолепное время, мы очень много сделали тогда. Были значительно впереди зарубежных работ. Но и сейчас в московском офисе Intel большинство – выпускники Физтеха. Они участвуют в выполнении важнейших интеловских проектов. И в Америке в Intel «физтехов» много.

«Физтехи» уезжают, уезжают, уезжают и оседают по всему миру. Как-то сидим в Сингапуре, едим китайские пельмени и рассуждаем, где там можно работать. В Австралии появилась ставка, в Сеуле… и кто-то говорит: «Ну прямо как раньше в общежитии Физтеха». А.Н. Козырев.

[1] Справка. Сергей Алексеевич Лебедев – разработчик первых вычислительных машин в Советском Союзе и основатель советской компьютерной индустрии. С.А. Лебедев внес основополагающий вклад в становление и развитие вычислительных наук в бывшем СССР. Им разработаны главные принципы построения и структура универсальных электронных цифровых вычислительных машин, организована работа коллективов разработчиков высокопроизводительных ЭВМ, промышленное производство этих ЭВМ и их внедрение, подготовка кадров. С.А. Лебедева называют "отцом вычислительной техники" в СССР.

В 1951 году я поступил в МФТИ, а в 1954 году я сделал свою первую важную студенческую работу. Я разработал арифметику с сохранением переноса (carry save arithmetic) — умножение, деление и извлечение корня квадратного без последовательных переносов и в 1955 году доложил о ней на открытой конференции. Первая западная публикация по этой теме появилась только в 1956. Сергею Алексеевичу очень понравилась моя работа. Этот метод один из двух, второй метод – умножение на несколько разрядов – предложил Робертсон (J.E. Robertson). Он в 1958 году был в Москве, и мы с ним встречались. С тех пор до настоящего времени эти два метода умножения и деления, один мой, второй Робертсона, составляют основу всей вычислительной арифметики. Конечно, весь мир не от меня знает это, но мое решение было публично доложено на конференции в Физтехе раньше всех. До сих пор никто ничего не сделал лучше. Как можно говорить, что Запад нас обогнал. Мы реализовали технологию полностью защищенных вычислений со строгой функциональностью в 1978 г., чего Запад не смог сделать до сих пор.

С.А. Лебедев

Расскажу историю с Лебедевым. Как я уже сказал, в 1951 я поступил в МГУ на физфак, а в 1952 сразу после первого курса мы с физфака МГУ (ФТФ МГУ) пришли на практику в ИТМиВТ. Я хорошо все помню, БЭСМ-1 еще не работала[2], у пульта сидел Сергей Алексеевич Лебедев, он всегда сидел на стуле, поджав одну ногу под себя, это его любимая поза. И он нам сказал тогда – эта БЭСМ еще не работает, а следующую машину мы будем делать вместе. И действительно, следующую машину мы вместе делали. Была линия БЭСМ, а была линия для систем реального времени, одна из этих машин М-40[3].

Ее делали мы. Кстати говоря, М-40 – это первая машина, где была реализована моя быстрая арифметика (carry save arithmetic). ИТМиВТ разработал М-40 в 1957 году. Это ламповая машина, но именно там была использована разработанная мной арифметика. Арифметическое устройство в то время составляло основную часть машины. И в 1961 году эта машина участвовала в первом в мире успешном противоракетном испытании.

Расскажу один эпизод, это может быть интересно. Как работала эта ламповая машина? Мы утром приходили, включали ее, находили приблизительно 40 неисправных блоков, заменяли их, после чего машина начинала работать. На озере Балхаш около поселка Сары-Шаган размещался полигон для отработки системы противоракетной обороны, а ракеты запускали с Волги (с полигона «КапЯр» – Капустин Яр). Целый день мы готовились, проводили тесты. И вот объявляется 1-часовая готовность, 5-минутная готовность – все великолепно, все готово. Центральное управление дает пуск. Баллистическая ракета запущена. Она летит 12 минут. Как только ракета полетела, почти мгновенно (через несколько секунд или через одну минуту) взорвалась лампа. Я все время смеюсь, что сейчас найти ошибку в кристалле очень трудно. А тогда было легко, взорвалась лампа, мы тут же ее заменили. А ракета летит …. Тесты все провели, и все, заработала машина, мы сбили ракету. Вывели данные, тогда дисков не было, просто печать на бумагу, и тут взорвалась вторая лампа. Если бы мы не успели вывести данные на печать, то этот результат пуска пропал бы. Вот такая это была надежность. Поэтому делая следующую ЭВМ 5Э92б – она была полупроводниковая – мы уже сильно озаботились надежностью.

Мы сделали машину, в которой все одиночные сбои обнаруживались аппаратурой, и машина автоматически перезапускалась. Машина оказалась довольно надежной, в среднем только 4-8 отказов в год. Это было не особенно нужно, но мы отработали полностью эту технологию абсолютно надежной машины. В то время в СССР многие коллективы делали разные машины, а потом стали копировать американские ЭВМ серии IBM/360, что не очень хорошо. Расчет был на то, что можно будет использовать зарубежное программное обеспечение — и в СССР наступит расцвет вычислительной техники. Этого, конечно, не произошло. Потому что после того, как все разработчики были собраны в одно место, творчество закончилось. Нужно было просто угадать, как сделаны западные, в действительности, уже устаревшие вычислительные машины. Передовой уровень известен не был, передовыми разработками не занимались, была надежда на то, что хлынет матобеспечение …. Вскоре стало ясно, что матобеспечение не хлынуло, заимствованные куски не подходили друг к другу, программы плохо работали. Все приходилось переписывать, а то, что доставали, было древнее, плохо работало.

Сергею Алексеевичу предлагали копировать IBM/360, он отказался и резко выступил против такого варианта в пользу разработки ЭВМ четвертого поколения совместно с английской фирмой ICL. Свою позицию он изложил на совещании в Минрадиопроме в декабре 1969 года.

Все наши успехи – это фактически потрясающий результат этого гениального решения Лебедева.

---

[2] http://www.computer-museum.ru/histussr/29-3.htm

[3] Описание и технические характеристики см. http://www.computer-museum.ru/histussr/m40.htm

«Система IBM/360 — это ряд ЭВМ десятилетней давности. Создаваемый у нас ряд машин надо ограничить машинами малой и средней производительности. Архитектура IBM/360 не приспособлена для больших моделей (супер-ЭВМ). Англичане хотят конкурировать с американцами при переходе к ЭВМ четвертого поколения. Чем выше производительность машины, тем в ней больше структурных особенностей. Англичане закладывают автоматизацию проектирования. Система математического обеспечения для "Системы-4" динамична, при наличии контактов ее вполне можно разработать. Это будет способствовать подготовке собственных кадров. Их лучше обучать путем разработки собственной системы (совместно с англичанами)». Из стенограммы совещания 1969г.

И все же победила другая точка зрения, приведшая нашу страну ко все большему отставанию в области вычислительной техники, а Сергей Алексеевич стал разрабатывать вычислительные комплексы (Эльбрус). Это был выдающийся человек.

Эльбрус, или Эль-Борроуз

В середине семидесятых годов прошлого века, когда создавался Эльбрус-1, среди разработчиков программного обеспечения была в ходу шутка, когда вместо Эльбрус говорили Эль-Борроуз. Автором такого произношения был Лев Николаевич Королев – основной разработчик операционной системы для БЭСМ-6.

ЭВМ Эльбрус-1

Когда мы начинали разрабатывать первый Эльбрус, мы не думали о безопасности вычислений, Такой проблемы тогда не существовало. Мы занялись строгой функциональностью архитектуры, чтобы поддержать языки высокого уровня. В то время, в 72-ом году, считалось, что программирование высокого уровня обеспечивалось такими языками, как Algol и Fortran, которые были вынуждены ориентироваться на существующие архитектуры для обеспечения достаточной производительности вычислений. Следовательно, чтобы поддержать существующие языки в новой машине, нам фактически пришлось бы ориентироваться косвенно, через языки, на существующие архитектуры. Это было бы в корне ошибочно.

В то время уже существовали машины с поддержкой языков высокого уровня, но базировались они на этом ошибочном подходе. Например, фирма Burroughs выпускала машину, исполнявшую расширенный Алгол (Extended Algol). Типы данных реализовывались через теги, хранящиеся в памяти вместе с данными. Использование тегов было хорошей идеей, и мы позаимствовали её у Burroughs. Но то, как они использовали эти теги, было просто недоразумением. С помощью тегов Burroughs поддерживала на аппаратном уровне статические типы, что можно делать (и делается) в значительной степени и без тегов. В Burroughs это работало так: каждой используемой ячейке памяти жестко приписывался некоторый тип. Эта информация применялась для автоматического преобразования данных во время их записи в память. Если, например, вещественное число сохранялось в область памяти, отмеченную тегом целого числа, то машина динамически превращала вещественное в целое перед тем, как осуществить запись. Это соответствовало семантике Алгола, ориентированного на старые архитектуры.

Функционально правильное решение, исполненное в Эльбрусе, связывает тип данных, описываемый тегом, не с памятью, где эти данные находятся, а с самими данными. В любую ячейку памяти (или в регистр) могут записываться данные любого типа вместе с тегом, описывающим тип этих данных.

Это единственно правильное решение (динамические типы данных), и оно реализовано в Эльбрусе.

Burroughs – это гениальная фирма, они сделали очень много нового, но мы их идеи значительно развили. Они придумали теги, они первыми сделали дескриптор, стек, очень много всего, но они совсем не обеспечили защищенность. Они создали объектно-ориентированные языки программирования. Мы много с ними взаимодействовали, и взаимопонимание у нас было полное. Конечно, мы их опередили, они сделали машины Burroughs, которые, не обеспечивали защищенность вычислений. Тогда это не очень важно было. А мы сделали машину, которая была быстрой и имела полную защищенность. Это значительно более мощный результат. Но надо отдать должное Burroughs, они пионеры, они гораздо раньше многое получили, но они фактически не довели дело до конца. Поддержав существующие статические языки, Burroughs, по сути, поддержала старую аппаратуру, потому что именно на неё были ориентированы статические языки. Разрабатывая новую архитектуру, Burroughs в итоге ориентировалась на старые машины. К тому же теги можно было менять на ходу в непривилегированном режиме,

что тоже большая ошибка. Я еще раз повторю, Burroughs – это выдающаяся фирма, так как они – пионеры, но мы осуществили дальнейшее развитие и реализацию этих идей, они это открыто признавали. Мы с ними активно работали все время. Справедливости ради, надо отметить, что динамическое приведение типов данных во время их записи в память далось Burroughs почти бесплатно. Для такого преобразования необходимо было при каждой записи в память узнавать тип ячейки, статически определённый на языке типа Алгол, в которую сохранялись данные. Это нужно было для того, чтобы узнать, каким образом данные должны быть преобразованы. Получается, что каждая запись в память требовала предварительного считывания. Такие лишние чтения сейчас при современной технологии реализации памяти выглядят ужасным расточительством. Но дело в том, что тогда использовалась ферритовая память. Перед тем, как писать в такую память, её нужно было размагничивать. А размагничивание делалось посредством чтения. Поэтому считывание типа ячейки во время записи не приводило к дополнительным накладным расходам. Оно просто совмещалось с размагничиванием.

В Эльбрусе реализована полная защита вычислений, что теоретически является неулучшаемым результатом. Причем это признано всеми ведущими зарубежными фирмами, которые занимаются защищенностью, в том числе Макафи (McAfee) и другими профильными фирмами. Мы с ними контактируем, они очень высокого мнения о нашей технологии. Я, конечно, понимаю, что защищенность очень трудно внедрить, потому что для этого нужно теоретически изменить текущую совместимость. Но мы достигли этого «теоретического» результата. И это выдающийся результат. Ведь мы выпустили первый Эльбрус в 1978 году, когда никому защищенность не была нужна. Машины стояли в отдельной комнате, закрывающейся на ключ. Приходил человек, ставил диски со своими магнитными лентами, начинал работать, кончал работать, машина стояла, он снимал свои диски и уходил, другой приходил. Какая там защищенность? Никому это не нужно было. Нужна была простота программирования, и мы её достигли. У нас защищенность была потому, что была абсолютно чистая логически архитектура, и обеспечение защищенности было просто побочным эффектом этого теоретически чистого подхода, что наиболее убедительно доказывает его теоретическую чистоту и правильность. Получился очень мощный результат. Сейчас же с защищенностью большие проблемы. Сейчас фактически не создают систему защиты вычислений, а реагируют на действия хакеров. Как это происходит? Хакеры придумывают что-то, а бороться против каждой новой хакерской разработки приходится все сложнее и сложнее. Производительность хакеров «теоретически» не ограничена. А мы ничего специально для защищенности не делали. У нас очень простая реализация и полная защищенность.

И все это потому, что основы были правильные. Принципы были правильные. Есть арифметика, есть какие-то операции, и есть типы данных. Бывают разные операции, например, есть указатели, которые смотрят в память, с ними нельзя производить арифметические операции. Логически это не нужно. И наша машина просто смотрит, если это указатель, то для него только операции с указателями разрешаются, если это целочисленная переменная, то только целочисленная арифметика позволена. У нас машина знает типы каждого числа (теги) и может контролировать правильность типов аргументов каждой операции. А в современных машинах над указателями можно совершать любые операции, что реально никому не нужно и очень опасно с точки зрения возможности нарушить защищенность вычислений.

Мы разработали технологию защищенных вычислений в 1978 году. Intel попробовал в том же направлении двигаться, и в 1981 году они разработали микропроцессор iapx 432, машина называлась Intellec. К тому времени у нас уже 3 года работал Эльбрус 1. Тогда, как вы знаете, все копировали машины, кроме нас, поскольку ЦК и Совмин ориентировались на копирование западных машин. В 1981 году Правительство обратилось к ученым, в том числе ко мне с вопросом, – не нужно ли копировать процессор iapx 432? Так как я был защитником идеологии type safety, меня включили в комиссию. И я написал тогда большую бумагу с критикой интеловского процессора. Написал, что машина плохая, и что через несколько месяцев она провалится. И она провалилась. Когда я пришел в Интел, то нашел статью Боба Колвелла (Bob Colwell), того гениального человека, который в 1995 году создал P6, один из первых на Западе суперскаляров. Он анализировал машину iapx 432 и пришел к тому же выводу, что и я: идея правильная, но реализация неверная. В статье Колвелла есть конкретные цифры. Например, вход в процедуру занимал до пятидесяти обращений к памяти. Как можно было на такой машине работать? Я знал ее основу, знал, что она неправильно была сделана, то есть все данные были представлены с их типами. В Эльбрусе к каждому слову есть несколько разрядов, которые говорят, какого оно типа: целое, вещественное, переменная и так далее. А они сделали так, что в массиве все типы данных должны быть одинаковые, и типизация должна была присваиваться всему массиву. У меня спросили, а нужно ли копировать эту машину? Я ответил, что уровень проектирования этой машины крайне низкий, и, конечно, её не следует копировать. Через несколько месяцев она провалилась с треском. Руководство Интела – в лице Гордона Мура (Gordon Moore) – сказало: хватит этой научной работы, x86 forever (x86 навсегда!), – и это решение нанесло колоссальный ущерб и вычислительной технике вообще, и всем работам по защищенности, в частности. Все сразу во всем мире прекратили все работы по защищенности. Если Intel не справился, кто же тогда рискнет? И Intel до сих пор с x86. На этом примере у нас тоже преимущества колоссальные, мы были гораздо более просвещенными, чем Intel.

В Эльбрусе просто на самом нижнем уровне контролировалась правильность работы с типами данных, и этого было абсолютно достаточно. Все просто, полная защищенность обеспечена простыми методами. Мы обсуждали эту тему со специалистами из Макафи (McAfee), они очень высокого мнения о наших результатах. Они говорят: у вас железо и операционная система – это святыни. С ними хакеры абсолютно ничего не могут сделать. Но вот какая беда. Сейчас 80% проблем с защищенностью связаны с тем, что хакеры знают ошибки в пользовательских программах и пользуются ими, например, в банковских программах. Против этого есть логически очень простой метод – доказательство правильности программы. Если программа очень критическая, надо применять доказательство правильности её работы. Это хорошо разработанная технология. Но в современных машинах она бессмысленна, её сейчас невозможно применять, из-за того, что архитектура плохая. Хакеры что делают? Пользовательскую программу сначала ломают хитрыми методами, пользуясь несовершенством архитектуры, потом поломанную начинают использовать. А в Эльбрусе это совершенно невозможно. Там строгая межпроцедурная защита, препятствующая порче работы процедур. Поэтому только в Эльбрусе можно эффективно использовать доказательства правильности, а они все эти 80% снимают. То есть это – уникальные явления, нигде еще до сих пор не догнали нас в этом совершенно. Все думают, что мы отстали навеки, но Западу далеко до нас, просто далеко. Так что Burroughs – это очень хорошо, я очень уважаю Burroughs. Аналогичная ситуация сейчас происходит с NVIDIA. Существующие машины, к сожалению, далеко не универсальные с точки зрения производительности, а то, что мы сейчас делаем, абсолютно универсальные машины. Вот я начал говорить про NVIDIA, это такой же прорыв в решении проблемы параллельности вычислений, какой в свое время сделала фирма Burroughs в смысле функциональности. Но архитектура NVIDIA – это не универсальное решение. Наша задача – полностью решить проблему эффективности **универсальных** параллельных вычислений.

Языки высокого уровня

Работая над Эльбрусами, мы пришли к тому, что язык высокого уровня – это динамический язык со строгим контролем указателей. К такому же выводу, независимо от нас, пришел Никлаус Вирт (Niklaus Wirth) – человек номер один в языках программирования. Он тоже понял, что программирование высокого уровня строится на type safety (типовая защита) и динамике типов. Вирт создал динамический по типам язык Эйлер (Euler). Все были просто в восторге от этого языка. Он был невероятно интересным, но ему недоставало эффективности, так как он не поддерживался аппаратурой. Требовался избыток программного динамического контроля. Тогда Никлаус Вирт принял компромиссное решение. Он сказал: «Да, типы должны быть, но пусть они будут статическими. И контроль должен быть… но, возможно, не очень сильный, поскольку, например, контроль выхода за границу массива требует дополнительных команд».

Мы же с самого начала подразумевали, что типы проверяются аппаратно. Например, если у вас есть указатель на массив, то никто не может, имея этот указатель, выйти за границы массива. Это технически невозможно. В результате мы сделали настоящий язык. У нас были теги, то есть ко всем данным добавлялись несколько разрядов, описывавших тип. Мы сделали контроль указателей. Одним словом, мы создали язык программирования Эль-76. Автором и основным разработчиком был Владимир Пентковский (к сожалению, он скончался в декабре 2012 г).

На Эль-76 была написана операционная система Эльбруса. Разработкой руководил Сергей Семенихин[4]. В середине 70-х годов всего 26 человек сделали мультипрограммную, мультипроцессорную, мультитерминальную операционную систему. В то время везде использовался в основном пакетный режим. А у нас такая развитая операционная система! К тому же наш ти́повый подход резко упрощал программирование. Он даже увеличивал производительность и при этом не требовал введения привилегированного режима даже в операционной системе. Мы три поколения машин на этом подходе сделали: первый, второй и третий Эльбрус.

[4] Справка. Семенихин Сергей Владимирович. Учредитель АО «МЦСТ», заместитель генерального директора по науке – направление «Операционные системы», Доктор технических наук, профессор, Лауреат Ленинской премии СССР, Заслуженный деятель науки РФ, Награжден орденом Трудового Красного знамени. Работает в МЦСТ с 1996 г.

Наши машины использовались в очень ответственных системах. Противоракетная оборона Москвы, контроль космоса, атомные проекты в Арзамасе. При этом все наши пользователи говорили, что отлаживать на нашей машине раз в 10 быстрее, чем на старых машинах. Впечатление было такое, будто работаешь с постоянно включенной отладочной системой, причем без потери эффективности. Когда наши ребята получили западные машины – в начале 90-х годов – они ко мне стали подходить и говорить: как на этих машинах можно работать? На них нельзя отлаживать программы. Ужасно!

И что же стало с этим подходом, если смотреть исторически? Мы выпустили первую машину в 1978 году. Тогда же прошел первый тест по операционной системе. До начала 90х годов в нашей стране широко использовался Эльбрус 1, 2. Примерно в это же время во многих университетах мира шли исследовательские работы в области type safety. Как я уже говорил, в компании Интел (Intel) тоже заинтересовались этим подходом, они выпустили машину с type safety на базе процессора iapx 432, она называлась Intellec. Это была драматически неудачная машина. Она обеспечивала type safety. Но там были элементарные ошибки.

В этой машине была реализована защита указателей, что конечно же, очень правильно. Однако типы давались не каждому элементу массива, а всему массиву (все массивы должны были содержать данные одного и того же типа). При входе в процедуру, у операционной системы запрашивалось четыре сегмента! Два сегмента для параметров: пойнтерный и скалярный. И два сегмента для локальных данных: тоже пойнтерный и скалярный.

После провала iapx 432 весь мир бросил заниматься аппаратной поддержкой языков высокого уровня. Все считают, что это красивая идея, однако непрактичная. Считают, несмотря на то, что у нас была полная реализация этой идеи, причём реализация эффективная.

Сейчас все машины – суперскаляры

Все современные машины – это так называемые суперскаляры. Самые первые машины до 1995 года работали иначе, не считая Эльбруса, он и в этом смысле был самый первый (первый в мире суперскаляр – Эльбрус 1 (1978 г.)). Посмотрите, какие были все старые машины и что сейчас на входе. Одна операция, потом другая, все последовательно. Когда я в свое время сделал арифметику, в машине было только одно устройство – арифметика. Память и арифметика – и больше ничего. Поэтому такое последовательное представление программы было нормально, просто управление счетом – это не описание алгоритма, это было управление единственным устройством. Так это было до начала 70-х годов, а потом стало много оборудования. И теперь последовательность нужно было переставить в процессе счета в параллельное выполнение, а переставить – это не так просто. Нужно было угадать, что переставить, чтобы эти команды не мешали друг другу. Чтобы не нарушать совместимость, это можно было делать явным образом, транслятор, локальный для каждого вычислительного модуля, может это делать до начала счета. Тогда на вход машины программа будет представлена уже в распараллеленном виде, а на вход локального транслятора в совместимом последовательном виде. Вот Эльбрус 3 и в этом смысле впереди всех был, благодаря двоичной трансляции, которая вводилась один раз до начала счета.

Сейчас все машины – это так называемые суперскаляры. Машина сама берет последовательное представление и превращает его динамически в параллельное. Первая суперскалярная машина в мире – это Эльбрус-1. Мы его сделали в 1978 году. Но я хотел бы отметить, что после Эльбруса-2 мы сказали, что это несовершенная архитектура, а все сейчас в мире выпускают именно такую суперскалярную архитектуру. Это очень сложная архитектура. Потому что машина сама динамически распараллеливает и выполняет вычисления, а распараллеливать можно только маленький отрезок последовательного кода, который виден, а это плохо, так как при этом используется небольшая часть всего параллелизма. Мы сейчас разработали новую архитектуру, где даже для последовательной программы есть внутреннее математическое обеспечение, локальное в машине, оно распараллеливает её до начала выполнения, результат счета в три - четыре раза быстрее. А весь мир пока живет на суперскаляре, от которого мы отказались в 1985 году. Кто может сказать, что мы отстали? У нас отстала электроника и способность продавать, а в архитектуре мы намного впереди Запада.

Возникает естественный вопрос о доказательности правильности, которая отметает 80%. Не совсем понятно, что это такое. Неспециалист слышит знакомые слова, но что за ними? Неволин И.В.

Это хорошо разработанная технология. Не я в ней главный специалист и не рискну объяснять, но она хорошо разработана, вы ее описание можете найти. Для каждого алгоритма, для каждой процедуры вводится такая формула конечного результата, как правило, машина что-то считает и по результатам определяется ее корректность. Есть программа, а есть ее абстрактное описание. И есть метод, который это доказывает. Я могу рассказать один интересный факт. Мы связались с фирмой NICTA из Австралии, специализирующейся на защищенности вычислений. Проблема защищенности – доказательство правильности. Они взяли и доказали правильность ядра Linux. Что они сделали? Чтобы никто не попортил это ядро, они окружили его, как я смеюсь, «берлинской стеной». Связь с ним только по месседжам, т.е. просто нельзя связаться, а это замедляет вычисления. Но ядро стало защищённым, его нельзя было

испортить и им можно было пользоваться. Они много времени потратили и доказали правильность ядра теоретически. Но это непрактично и не универсально.

Сейчас среди специалистов считается, что доказательство правильности – это красивая вещь. Но на современных машинах не стоит этим заниматься. На самом деле это верно только для современных машин. Для Эльбруса и для будущих машин, которые сейчас разрабатываются нами, это уже неверно. Так как в этих машинах, благодаря технологии защищенности, все процедуры защищены друг от друга, их нельзя испортить.

Параллелизм

А сейчас по поводу параллелизма. В последние годы суперскаляр практически остановил развитие универсальной компьютерной архитектуры – рост её производительности. Согласно требованию программной совместимости суперскаляр использует строго последовательный код программы, представляющий алгоритм для вычисления. Однако для повышения производительности необходимо уметь эффективно использовать большое количество аппаратных ресурсов, которые могут и должны работать параллельно. Суперскаляр – довольно старая архитектура, разработана она давно, когда такой большой объём параллельного оборудования не был доступен. Поэтому сейчас суперскаляру перед тем, как выполнять вычисления (что надо делать сейчас с большим уровнем параллельности), необходимо распараллелить последовательно представленный входной алгоритм. И естественно, эту совсем не простую работу необходимо делать не медленнее, чем будет считаться такая распараллеленная программа.

Более того, эта совсем не простая работа значительно усложняется ещё, как минимум, двумя обстоятельствами.

В программах много условных переходов. Во время процесса распараллеливания довольно часто значение предиката, по которому будет выполняться переход, ещё неизвестно. Поэтому необходимо либо распараллеливать обе веточки вычислений (но будет исполняться только одна), и это ещё дополнительная работа, либо «угадывать» будущее направление перехода и терять время, если это «угадывание» было ошибочным.

Другая проблема – это распараллеливание считываний и записей в память.

Сейчас совершенно очевидно, что суперскаляр – это очень «древняя» архитектура. Отечественные разработчики Эльбруса, которые реализовали суперскаляр раньше всех, когда аппаратуры было ещё недостаточно для высоко параллельных вычислений, отказались от суперскалярной архитектуры в 1985 году, начав проектировать Эльбрус-3 – не суперскаляр. И этот шаг – тоже первый во всей вычислительной технике, не повторенный ещё никем.

В этом смысле легко объясним успех NVIDIA, они занимались разработкой графических процессоров, которые с самого начала были высоко параллельными и не требовали совместимости. Поэтому в своей работе NVIDIA не использовала суперскалярную архитектуру с самого начала, и им не пришлось от нее отказываться.

Они не используют последовательное представление программы, они разрабатывают графику заведомо параллельную с самого начала, поэтому у них отличные результаты. Но это не универсальные машины. Мы проектируем примерно такую же параллельную архитектуру, не хуже, но наше решение универсальное. Мы ускоряем не только «циклы», как NVIDIA, но и скаляр.

NVIDIA, так же как и Burroughs, реализовала только базовые компоненты целевой архитектуры. Burroughs реализовала только элементы защищённости, но не защищённость в целом, а NVIDIA реализовала только параллельность циклов, но без поддержки универсального параллелизма (включая скаляр).

Мы реализовали полную защищённость и работаем над реализацией полного универсального параллелизма (включая скаляр). Эта архитектура будет по производительности универсальной – она будет высокопроизводительно вычислять широчайший спектр различных алгоритмов, включая алгоритмы, реализующие Искусственный Интеллект, поддерживающие цифровую экономику и т.д.

Сейчас состояние всей компьютерной технологии критическое

Какие эффекты и последовательности развития компьютеров, проявившие себя в последние десятилетия, оказались наиболее неожиданными и парадоксальными? Здесь я могу сказать, что сейчас состояние всей компьютерной технологии критическое, причем по очень простой причине. Здесь я хочу повторить очень важную информацию, приведённую раньше.

В старой машине было одно исполняющее устройство, чисто последовательное представление программы. Теперь, когда устройств стало много, решили сохранить совместимость, оставили на входе последовательное представление, а внутри стали распараллеливать программу. Для машины, которая была сделана в 1995 году, это было оправдано. Мы в 1978 году так сделали, а в 1985 году отказались от этой технологии. На Западе первую такую машину выпустили только в 1995 году, и тогда это было обоснованно, потому что кремниевая технология была не очень хорошая. И сейчас, когда кремниевая технология уже стала совершеннее, она позволяет сделать в пять-десять раз больше исполняющих устройств каждого типа, тесно связанных между собой. Последовательная архитектура такого не могла сделать, поэтому ввели частичный параллелизм. Сначала мы, потом они использовали неявный параллелизм. В машине уже было много оборудования, она задание получала последовательно, сама его перед выполнением распараллеливала. Но сильно не распараллелишь. Когда оборудования не так много, такого распараллеливания было достаточно, все это оборудование использовалось хорошо. И это было где-то до начала 2000-х годов. А уже с 2000-х годов оборудования стало гораздо больше, и аппаратное распараллеливание не могло полностью использовать его. Мы это предвидели уже в 1985 году и отказались от суперскаляра с ограниченным распараллеливанием, мы стали работать над системой с явным представлением параллелизма. Теперь оборудования много, сейчас на один кристалл помещается 30-40 ядер. Больше доступного оборудования одно современное ядро уже не может использовать по вышеизложенной причине. То есть, сейчас архитектура может использовать всего 2-3% доступного оборудования. Ужасное состояние. Это катастрофа в архитектуре. Эта катастрофа повела за собой катастрофу в языках, поскольку языки непрактично делать высоко параллельными, когда аппаратура все равно почти последовательная. Так нельзя, все языки испорчены. Разработчики языков делали так, чтобы их языки хорошо выполнялись на этой испорченной архитектуре. Поэтому они никакие революционные черты в языки не вводили. Все языки искалечены, операционная система искалечена, все в ужасном состоянии. Сейчас нужно революцию делать.

Да, неожиданные эффекты, но сейчас такая ситуация. В 2003 году суперскаляр еще можно было оправдать, тогда он использовал весь кристалл. Сейчас в один кристалл помещается уже 30 процессоров. А NVIDIA использует этот кристалл, потому что там сама задача представляется параллельной с самого начала.

Разумеется, сейчас все ядра работают на разных задачах, они много десятков задач решают. Конечно, есть многоядерное программирование, это головоломка для программиста. Связь между ядрами очень медленная, программист голову ломает и начинает калечить большие задачи.

Используется искалеченное программирование. Практически во всех языках программирования представление параллелизма описываемой задачи не поддержано явно.

Кроме того, не поддержана полная защищённость вычислений, отработанная до конца ещё в архитектуре Эльбруса-1 (1978). В настоящее время трудно переоценить важность введения полной защищённости в нашу жизнь. Т.е. сейчас это уже всем стало ясно.

Раньше каждое новое поколение машин сильно ускорялось, в основном потому, что совершенствовалась кремниевая технология. Сейчас суперскаляр уже невозможно так ускорять. Увеличение на 5% считается заметным увеличением производительности. А мы здесь в Москве показали на нашем проекте в три раза больше скорость, потому что убрали этот последовательный вход в аппаратуру даже в совместимом варианте.

Сейчас все используют суперскалярную архитектуру, только NVIDIA делает иначе, но их архитектура не универсальная. А мы разрабатываем не суперскалярную, но универсальную архитектуру, которая даже в совместимом режиме показывает скорость в 2,5 -3 раза больше, чем суперскаляр. А в несовместимом режиме наша разработка не будет уступать NVIDIA по производительности.

Приведённый выше анализ истории убедительно показывает, что в нашей стране мы задолго предвидели эти критические ситуации в обеих фундаментально важных областях вычислительной техники:

- высокой функциональности, ответственной за полное решение проблемы защищённости вычислений и простоты программирования и
- высокой универсальной параллельности вычислений, ответственной за производительность, включающей и цикловые, и скалярные алгоритмы.

Мы предложили и реализовали (в части параллелизма – сильно продвинулись в реализации) фундаментальное решение этих проблем.

В чем заключается шанс России на технологический прорыв в области компьютеров?

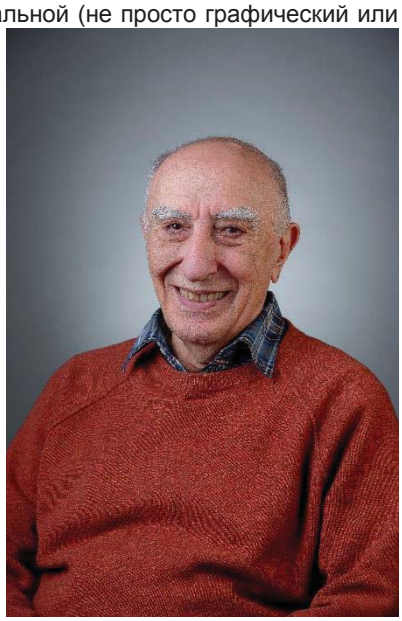
Шанс у нас есть, но это действительно трудный вопрос. В области архитектуры мы впереди всех, а современной кремниевой технологии у нас ещё нет, ну и что делать? В этой части сейчас можно было бы работать на технологии TSMC (Тайвань). Это открытое предприятие. Если бы Правительство могло выделить достаточно большое финансирование, но тут нужны миллиарды долларов, тогда можно было бы успешно реализовать все эти идеи.

С.А. Лебедевым определен подход к отечественному пути развития высокопроизводительной вычислительной техники, а именно: двигать ее самостоятельно, не копируя зарубежные образцы. Именно по этому пути идет коллектив АО «МЦСТ», вышедший из ИТМ и ВТ (в настоящее время возглавляемый

Александром Кимом), который продолжает развивать отечественную архитектуру универсальных высокопроизводительных микропроцессоров «Эльбрус». Несмотря на скромное финансирование, последние 8-ядерные микропроцессоры линии «Эльбрус», изготовленные на «отсталой» кремниевой технологии, сопоставимы по пиковой производительности с современными универсальными микропроцессорами фирмы Intel. По моему мнению, будучи поддержан экономически, этот коллектив вполне может осуществить технологический прорыв в области компьютеров.

Но имея свои собственные разработки, надо еще научиться их продавать. Это тоже немаловажно. Мы всех обогнали в архитектуре, я это показал, но остались большие проблемы в других областях.

Разумеется, если появятся такие машины, тогда пользователям придется все программы переписывать, поскольку все они будут несовместимы. Но здесь возникает труднейшая проблема – необходимо создать условия пользователям для перехода на новую совместимость. Пользователи должны видеть выгоды для такого перехода. При создавшейся критической ситуации в вычислительной индустрии есть естественный выход. Дело в том, что если отказаться от совместимости, то производительность машин будет в 10-20 раз больше, и в то же время машина будет универсальной (не просто графический или еще какой-либо ускоритель для суперскаляра), потому что кристалл будет почти весь работать над этой задачей. Тогда все-таки, несмотря на потерю совместимости, люди будут покупать ее из-за большой скорости. Кроме того, покупатели даром получат полную защищенность (security). В этой машине можно обеспечить и исполнение старых программ, там они тоже будут раза в 4 быстрее. И потом постепенно все перейдут на эту архитектуру. Скорость нужна всем. Для получения скорости люди будут переписывать программы.



Гуманитарный вопрос. Сейчас в некоторых институтах, университетах запустили систему оценок успешности науки, научных сотрудников. Как человеку и большому ученому вопрос: «Что Вы считаете лично критериями успешности научных сотрудников?» Ноакк Н.В.

Я считаю, что сильный ученый должен быть честным по отношению к технике. Он не должен защищать только то, что сделал сам. Если он увидел, что это неправильно, он должен изменить свое мнение. Тогда он действительно сильный. А если он что-то неправильное начинает защищать, значит это слабый человек.

Бабаян Борис Арташесович *Член-корреспондент РАН, директор по архитектуре подразделения Software and Solutions Group корпорации Intel, Intel Fellow.*

Ключевые слова

.

Babayan B.A. *Corresponding member of RAS, Director of architecture Software and Solutions Group (Intel), Intel Fellow.*

Keywords

Abstract

This article is formed from the statements of Boris Babayan and approved by him and therefore it may be served as its author's text. Most of the statements included in the article are taken from the audio recording of the conversation between Boris Babayan and a group of employees of the journal (A.N. Kozyrev, I. V. Nevolin, N.V. Noakk) 23.09.2018. Replicas of other participants of the conversation are inserted as insets in the text of the article. Also, the insets give information about the people and facts mentioned in this conversation without explanation. In preparation for this conversation, we have made a list of questions that are supposed to ask outstanding developers of domestic computing technology, as well as to study the earlier interviews and statements by Babayan, related to the history of the faculty of Cybernetics and MIPT. Hence the idea-to prepare an article for the journal Digital economy.

DOI: 10.34706/DE-2018-03-08

Приложение

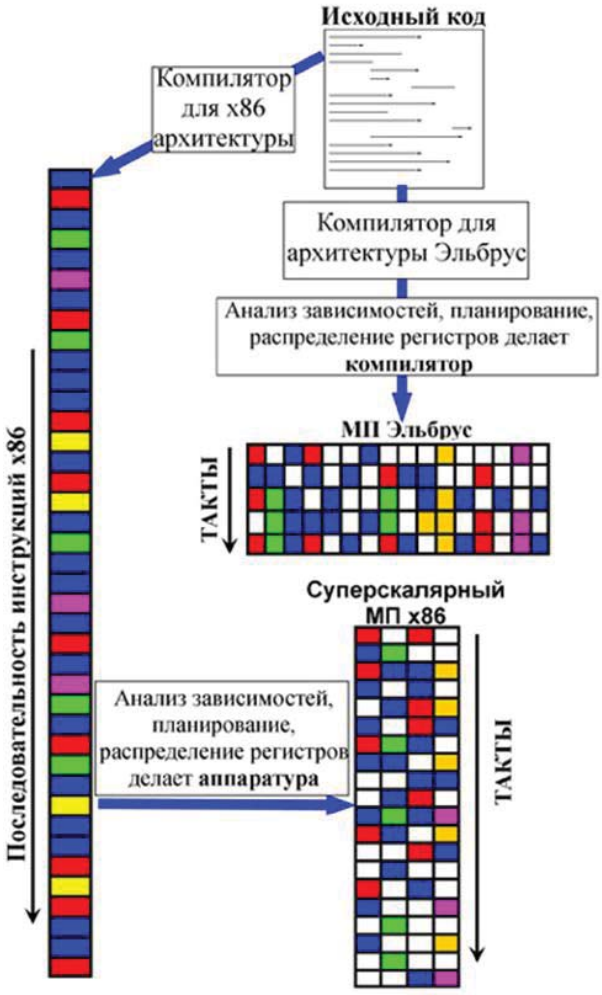
Краткое описание архитектуры Эльбрус

Работы над архитектурой «Эльбрус» начались в 1986 г. в коллективе Института точной механики и вычислительной техники (ИТМ и ВТ) им. С.А. Лебедева, в котором до этого были созданы советские высокопроизводительные комплексы «Эльбрус-1» и «Эльбрус-2». Разработка вычислительного комплекса «Эльбрус-3», которая велась под руководством Б.А. Бабаяна, была завершена в 1991 г. В этом вычислительном комплексе впервые были воплощены в жизнь идеи явного управления параллелизмом операций с помощью компилятора.

Начавшиеся с 1992 г. экономические изменения в России не позволили разработчикам «Эльбруса-3» завершить наладку комплекса. В том же 1992 г. коллектив разработчиков машин семейства «Эльбрус» выделился в компанию ЗАО «МЦСТ» и начал вести работы над микропроцессорной реализацией архитектуры «Эльбрус».

Архитектура «Эльбрус» – оригинальная российская разработка. Ключевые черты архитектуры «Эльбрус» – энергоэффективность и высокая производительность, достигаемые при помощи задания явного параллелизма операций.

Ключевые особенности архитектуры Эльбрус



В традиционных архитектурах типа RISC или CISC (x86, PowerPC, SPARC, MIPS, ARM), на вход процессора поступает поток инструкций, которые рассчитаны на последовательное исполнение. Процессор может детектировать независимые операции и запускать их параллельно (суперскалярность) и даже менять их порядок (внеочередное исполнение). Однако динамический анализ зависимостей и поддержка внеочередного исполнения имеет свои ограничения: лучшие современные процессоры способны анализировать и запускать до 4-х команд за такт. Кроме того, соответствующие блоки внутри процессора потребляют заметное количество энергии.

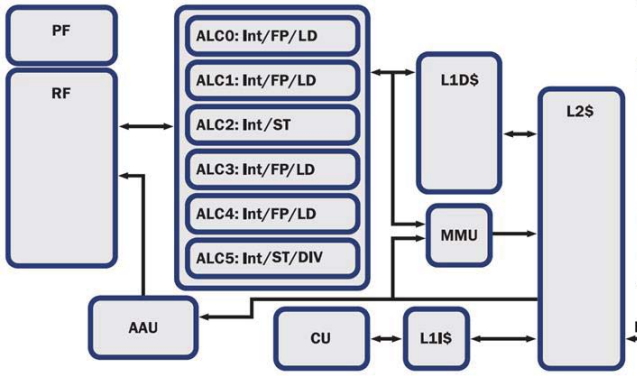
В архитектуре «Эльбрус» основную работу по анализу зависимостей и оптимизации порядка операций берет на себя компилятор. Процессору на вход поступают т.н. «широкие команды», в каждой из которых закодированы инструкции для всех исполнительных устройств процессора, которые должны быть запущены на данном такте. От процессора не требуется анализировать зависимости между операндами или переставлять операции между широкими командами: все это делает компилятор, исходя из анализа исходного кода и планирования ресурсов процессора. В результате аппаратура процессора может быть проще и экономичнее.

Компилятор способен анализировать исходный код гораздо тщательнее, чем аппаратура RISC/CISC процессора, и находить больше независимых операций. Поэтому в архитектуре Эльбрус больше параллельно работающих исполнительных устройств, чем в традиционных архитектурах, и на многих алгоритмах она демонстрирует непревзойденную архитектурную скорость.

Возможности архитектуры Эльбрус:

- 6 каналов арифметико-логических устройств (АЛУ), работающих параллельно.
- Регистровый файл из 256 84-разрядных регистров.
- Аппаратная поддержка циклов, в том числе с конвейеризацией. Повышает эффективность использования ресурсов процессора.

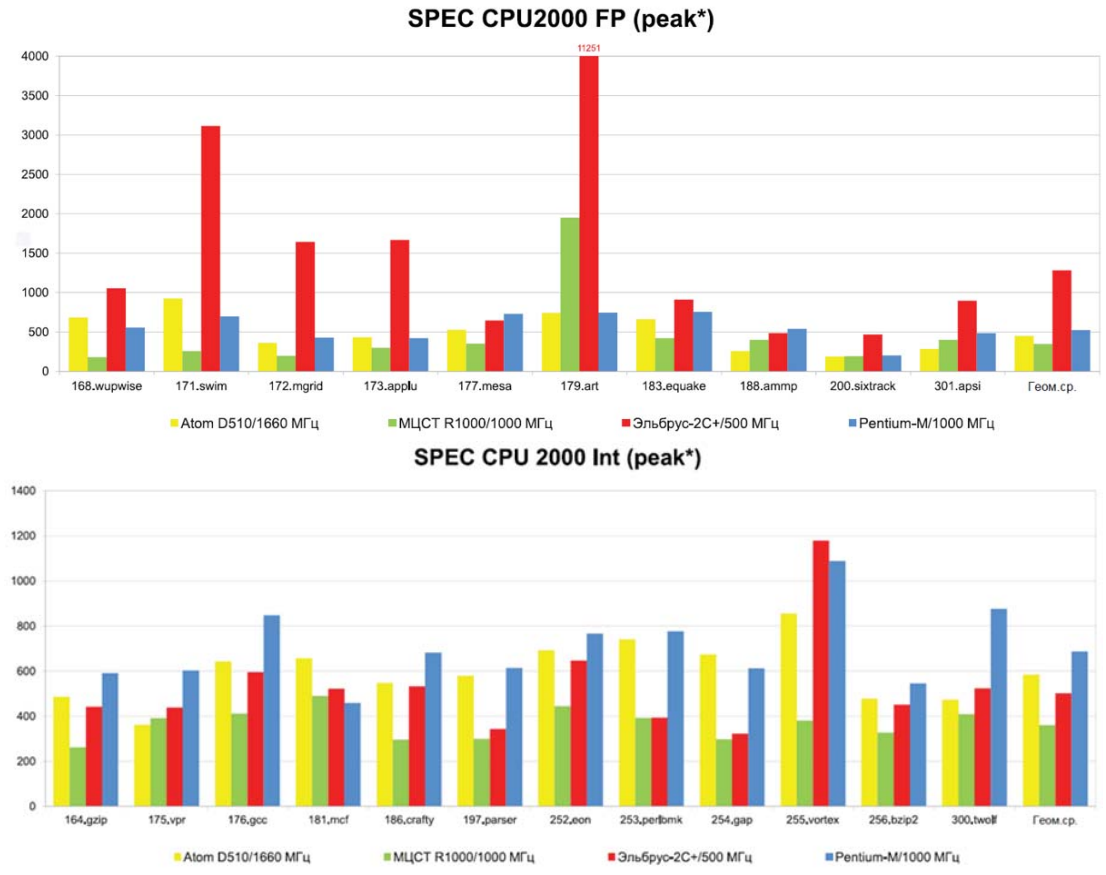
- Программируемое асинхронное устройство предварительной подкачки данных с отдельными каналами считывания. Позволяет скрыть задержки от доступа к памяти и полнее использовать АЛУ.
- Поддержка спекулятивных вычислений и однобитовых предикатов. Позволяет уменьшить число переходов и параллельно исполнять несколько ветвей программы.
- Широкая команда, способная при максимальном заполнении задать в одном такте до 23 операций (более 33 операций при упаковке операндов в векторные команды).



Производительность на реальных задачах:

Ниже приведена производительность процессора Эльбрус-2С+ на задачах из пакета SPEC2000 по сравнению с процессорами Intel Pentium-M ULV (1ГГц, кэш-память 1М, 2xDDR-266) и Intel Atom D510 (1,66 ГГц, кэш-память 1М, DDR2-800).

Производительность микропроцессоров Эльбрус-2С+ и МЦСТ R1000 на задачах



** Согласно требованиям SPEC Run and Reporting Rules, результаты тестирования Эльбрус-2С+ и МЦСТ R1000 публикуются как оценочные (estimates).*

Данные для Intel Pentium-M ULV получены с сайта spec.org, компилятор ICC 9.1. Для замера производительности процессора Intel Atom D510 использовалась собственная сборка тестов SPEC силами сотрудников МЦСТ.

Важно отметить, что правила комитета SPEC запрещают осуществлять модификацию исходных кодов тестов. Практика показала, что архитектура Эльбрус обладает значительным резервом производительности, который можно задействовать путём модификаций исходного кода в критических участках.

Эмуляция архитектуры x86

Обеспечивает исполнение ОС: MS DOS, Windows (95, NT, 2000, XP), нескольких вариантов Linux, FreeBSD, QNX

Еще на этапе проектирования МП Эльбрус у разработчиков было понимание важности поддержки программного обеспечения, написанного для архитектуры Intel x86. Для этого была реализована система динамической (т.е. в процессе исполнения программы, или «на лету») трансляции двоичных кодов x86 в коды процессора Эльбрус. Фактически, система двоичной трансляции создает виртуальную машину, в которой работает гостевая ОС для архитектуры x86. Благодаря нескольким уровням оптимизации удается достичь высокой скорости работы оттранслированного кода (см. диаграммы выше). Качество эмуляции архитектуры x86 подтверждается успешным запуском на платформе Эльбрус более 20 операционных систем (в том числе несколько версий Windows) и сотен приложений.

Защищенный режим исполнения программ

Одна из самых интересных идей, унаследованных от архитектур Эльбрус-1 и Эльбрус-2 – это так называемое защищенное исполнение программ. Его суть заключается в том, чтобы гарантировать работу программы только с инициализированными данными, проверять все обращения в память на принадлежность к допустимому диапазону адресов, обеспечивать межмодульную защиту (например, защищать вызывающую программу от ошибки в библиотеке). Все эти проверки осуществляются аппаратно. Для защищенного режима имеется полноценный компилятор C/C++ и библиотека run-time поддержки.

Даже в обычном, «незащищенном» режиме работы МП Эльбрус имеются особенности, повышающие надежность системы. Так, стек связующей информации (цепочка адресов возврата при процедурных вызовах) отделен от стека пользовательских данных и недоступен для таких вирусных атак, как подмена адреса возврата. Стоит отдельно отметить, что в настоящее время вирусов для платформы «Эльбрус» просто не существует.

Сфера применения микропроцессоров архитектуры Эльбрус	
Расширенный температурный диапазон, возможность локализации производства	Государственный заказ, промышленные компьютеры, автомобильная электроника
Повышенная защищенность от вирусных атак	Платежные терминалы, сетевые экраны, взломоустойчивые серверы
Высокая производительность на криптографических алгоритмах	Модули шифрования, защищенные тонкие клиенты, прочие системы безопасности
Высокая производительность на вычислениях с действительными числами (float, double)	Робототехника, авионика, промышленные контроллеры, системы обработки изображений, суперкомпьютеры
Работа под управлением бинарного компилятора в режиме совместимости с архитектурой x86	Интернет-терминалы, маломощные рабочие станции, малогабаритные настольные и встраиваемые компьютеры
Защищенный режим	Особо ответственные системы, отладочные стенды

Источник - сайт МСЦТ.