Суперкомпиляция и частичные вычисления до сих пор не вошли в широкую практику программирования. Почему и что делать?

Анд.В. Климов[1]

[1] *ИПМ им. М.В. Келдыша РАН*

Аннотация. Такие методы метавычислений как суперкомпиляция и частичные вычисления появились около полувека тому назад. С тех пор они плодотворно развивались с получением глубоких научных результатов, созданием программных прототипов, однако до сих пор не используются в практике программирования, хотя их авторы видели в них большую перспективу для автоматизации программирования и ожидали быстрого достижения целей. В чем же дело? Мы рассматриваем историю развития программирования с точки зрения эволюционной концепции метасистемных переходов В.Ф. Турчина, чтобы понять истоки методов метавычислений и их место в общей картине. Отмечаем, что в отличие от программных инструментов, легко входивших в практику, таких как, например, оптимизирующие компиляторы, эти методы не удалось реализовать так, чтобы их можно было использовать в автоматическом режиме, поскольку они содержат слишком много точек выбора решений, вариантов, степеней свободы, которые машина не может разрешить без помощи человека. Для дальнейшего продвижения к практике предлагаем уделить особое внимание погружению метавычислительных инструментов в современные среды программирования и разработке удобных человеко-машинных интерфейсов, где человек останется метасистемой высшего уровня, принимающей творческие решения. Кроме того, предлагаем включить суперкомпьютеры в арсенал инструментов разработчиков программ, как это произошло в прикладных инженерных областях, но почему-то разработка эффективных и надежных программ для критических областей применения не считается оправдывающей использование суперкомпьютеров.

Ключевые слова: метавычисления, метасистемный переход, суперкомпиляция, частичные вычисления, будущее программирования

Supercompilation and partial evaluation are still not widely used in practice. Why and what to do?

And.V. Klimov[1]

[1] *Keldysh Institute of Applied Mathematics of RAS*

Abstract. Such methods of metacomputation as supercompilation and partial calculations appeared about half a century ago. Since then, they have fruitfully developed, deep scientific results were achieved, many software prototypes were developed. However, they are still not used in the practice of programming, although their authors saw in them a great prospect for automating programming and expected to quickly achieve the goals. What's the big deal? We consider the history of programming from the point of view of the evolutionary theory of metasystem transitions of V.F. Turchin in order to understand the origins of metacomputation methods and their place in the overall picture. We note that unlike software tools that were easily put into practice, such as, for example, optimizing compilers, these methods could not be implemented so that they could be used automatically, because they contain too many points of choice of solutions, options, degrees of freedom that the machine cannot resolve without human assistance. For further progress to practice, we propose to pay special attention to embedding metacomputation tools in modern programming environments and developing and designing convenient human-machine interfaces, where a person will remain a top-level metasystem that makes creative decisions. In addition, we propose to include supercomputers in the arsenal of software developers' tools, as it happened in applied engineering fields, but for some reason the development of effective and reliable programs for critical applications is not considered to justify the use of supercomputers.

Keywords: metacomputation, metasystem transition, supercompilation, partial evaluation, future of programming

1. Введение

Мы называем *метавычислениями* область информатики (computer science & software engineering), где программы являются объектом глубокой обработки, анализа и преобразований. В нее входят частичные вычисления (partial evaluation) [**5**], суперкомпиляция (supercompilation) [**15**,**16**], частичная дедукция (partial deduction) [**10**] и другие методы, решающие задачи специализации, композиции, инверсии, распараллеливания, оптимизации, верификации программ и т.д. Бурно развивающиеся ныне мощные системы типизации (вплоть до зависимых типов, dependent type) также являются нетривиальной деятельностью над программами, и их тоже имеет смысл относить к метавычислениям. Объединение перечисленных задач и методов в одну категорию оправдано не только и не столько по этому простому признаку, но и потому, что в своем развитии они

сталкиваются с общими фундаментальными и техническими проблемами, решать которые надо на основе общего опыта и подходов.

Основатели области метавычислений — Валентин Ф. Турчин, Андрей П. Ершов, Йосихико Футамура (Yoshihiko Futamura), Нил Д. Джоунс (Neil D. Jones) — в 1970-80-х годах полагали, что методы метавычисления быстро войдут в практическое программирование и произведут революцию в программной инженерии. *Специализация* программ считалась первой задачей, которая должна изменить методы решения многих задач программирования. Проекции Футамуры [**2**] выглядели очень привлекательно для создания компиляторов из интерпретаторов и генерации эффективного специализированного кода из сборок универсальных компонентов.

С тех пор прошло почти полвека, но мы так и не видим эти методы в широкой практике, как ожидалось. Каждая из подобластей метавычисления постепенно развивается по своей логике и в соответствии со своими движущими силами, но ожидаемого общего прорыва не произошло и не происходит. Может создаться впечатление, что мы всего лишь продолжаем шлифовать методы и инструменты, изобретенные в прошлом веке.

Пришло время проанализировать причины, по которым развитие методов метавычислений застопорилось, если это действительно так; посмотреть, не ошибались ли мы с первоначальными предположениями; выявить, что недостаточно развито; прояснить препятствия; а также выбрать направления, что делать дальше для преодоления кажущейся или реальной стагнации в этой сфере; или сделать вывод, что ожидания были слишком высокими и всё и так идет хорошо.

Основные цели этого доклада:

1) напомнить представления об эволюции В. Турчина [**17**], служивших основаниями для ожиданий в 1970-е годы быстрого и плодотворного развитии методов метавычислений;
2) дать общую картину развития средств программирования через призму концепции В. Турчина, выявляя логику развития на пути к метавычислениям;
3) объяснить основания оптимизма «отцов-основателей» в том, что метавычисления необходимы, реальны и дадут большой практический эффект;
4) указать на «объективные трудности», общие для всех методов метавычислений, недостаточно осознаваемые в те времена (а именно, не удалось исключить человека как самой верхней метасистемы; автоматические системы не получились, а в лучшем случае — лишь автоматизация деятельности человека);
5) обсудить представления о том, как двигаться дальше, какого типа инструменты развивать и как двигаться к практике (а именно,

создавать методы и инструменты для деятельности человека как метасистемы высшего уровня).

Структура статьи следующая. Раздел 2 дает основные идеи концепции метасистемного перехода В. Турчина, нужные для понимания истоков и места метавычислений. Раздел 3 содержит взгляд с птичьего полета на развитие программирования через призму этой концепции как последовательности метасистемных переходов. В разделе 4 обсуждается появление суперкомпиляции как реализации замысла В. Турчина о метасистемном переходе над программами. В разделе 5 резюмируется нынешнее состояния дел в области метавычислений и в разделе 6 предлагается, по каким путям идти дальше. Раздел 7 содержит заключение с основными тезисами статьи.

2. Концепция метасистемных переходов В. Турчина и эволюция средств программирования

В. Турчин развивал методы суперкомпиляции, используя в качестве «руководства к действию» свою эволюционную теорию [**17**], равно применимую как к естественной эволюции биологического мира, человеческого общества, культуры, науки и технологий, так и к «искусственной эволюции», то есть сознательному выбору учеными и инженерами продуктивных направлений исследований и разработке подходов к получению новых результатов. Основное понятие этой теории — шаг эволюции, эволюционный скачок, называемый *метасистемным переходом* [**17**]. В. Турчин оттолкнулся от наблюдения, что во всех областях периоды медленного, слабо заметного развития чередуются с быстрыми скачками. Например, в науке это то, что Томас Кун [**11**] назвал *нормальной наукой* и *сменами парадигм* соответственно.

В. Турчин описал типовую структуру метасистемных переходов. В периоды медленного развития каким-то образом (случайным, методов проб и ошибок или сознательным решением исследователя или конструктора) из уже существующего материала выделяется, строится подсистема, называемая *управлением*. Ее функция — организовать, интегрировать системы предыдущего уровня, сделав их подсистемами нового уровня, обеспечить их размножение и совместное выполнение некоторых общих функций, обеспечивающих выживание (для биологических систем), плодотворность в решении новых задач (для культуры и научно-технического творчества) и т.п. Новая система, состоящая из управления и организованной совокупности подсистем, размноженных (с вариациями) из систем предыдущего этапа развития, называется *метасистемой*.

Характерное явление, признак, удобный для распознавания происходящего или произошедшего метасистемного перехода, — многократное размножение некоторых подсистем (как структурных — появление копий подсистем с каким-то вариациями, так функциональных

— некоторые виды деятельности становятся многократно повторяемыми, организованными в новый процесс ради новой цели). Заметив это, следует обратить внимание на механизмы, которые обеспечили размножение и единое функционирование; это и будет *управлением*, *управляющей подсистемой метасистемы*. В. Турчин назвал этот эффект *законом разрастания предпоследнего уровня.*

3. Примеры метасистемных переходов в программировании

Программируемый вычислитель. Появление программируемого вычислительного устройства в середине XX века. Здесь архитектура и система команд — управление, управляющая подсистема. Многообразие программ, исполняемых на одном устройстве — разрастание предпоследнего уровня.

Языки программирования. Появление языков программирования в 1950-е годы — сначала Фортрана, потом Алгола и Лиспа — метасистемный переход над программами. Язык и его реализация — управление. Повышение производительности программистов, увеличение числа программистов и числа программ — разрастание предпоследнего уровня.

Методы и средства реализации языков. Резкий рост числа языков программирования в 1960-70-е годы — разрастание предпоследнего уровня в метасистемном переходе над языками программирования. Языки стали привычным объектом создания, что, в частности, породило мечту построить «универсальный язык программирования» для всех применений и разработка кандидатов на эту роль (например, Алгол-68), что впоследствии была общепризнано невозможным. Здесь управление — методы и инструменты реализации языков; (само)применение языков высокого уровня для реализации себя и других языков. Разрастание предпоследнего уровня — рост числа языков и их реализаций.

Автоматизация создания языков. С 1960-х годов и особенно в 1970-е появление специализированных средств для описания языков (парсеров, «компиляторов компиляторов», метаязыков, макроязыков и макрогенераторов) — это метасистемный переход над средствами создания языков. Здесь управление — теория, методы и технологии создания языков; знания, как это делать. Разрастание предпоследнего уровня — увеличение числа инструментов создания языков, приведших к повышению их разнообразия, возникновение понятия предметно-ориентированного языка (domain-specific language, DSL) и идеи, что языков должно быть много, своих для различных областей применения. Ту волну DSL в значительной степени обеспечили макрогенераторы, которые впоследствии сошли на нет, но в 2000-е года мода на DSL возникла снова на новом уровне инструментов. (Отметим, что это поучительно и достойно более тщательного анализа).

Верификация программ. Идея и средства верификации программ на основе классической математической логики и логического вывода, развивающиеся с 1960-х годов, — это метасистема над программами, обеспечивающая другую деятельность по сравнению с эффективным исполнением: формальное доказательство корректности программы по отношение к выписанным утверждениям о ее семантике. Математическая логика и методы ее применения к языкам — здесь управление. Разрастание — появление некоторого числа систем верификации и, конечно, большого числа публикаций по этой теме. Эта область развивается до сих пор, но не заметна в практике так, как ожидалось. (Это хороший пример подумать, в чем дело.)

Системы типизации. Средства статической типизации в прошлом веке рассматривались как естественная часть собственно языка с прагматическими целями: для более легкого программирования и более эффективной реализации по сравнению со языками с динамической типизацией. Однако к концу XX века было осознано [**13**], что достаточно богатая система типизации, неброско называемая *зависимыми типами* (dependent types), оказалась не слабее языка математических утверждений вообще и о программах в частности. При этом коды программ сами выполняют роль математических доказательств, и нет необходимости в отдельном языке доказательств. Параллель между типизированными языками программирования и языком математики получила название *соответствия Карри-Ховарда* [**1**,**4**].[1] Богатую систему типизации естественно считать управлением над собственно программами как изображением алгоритмов (то есть без статической типизации), а типизированный язык — метасистемой над собственно программами. Здесь разрастание предпоследнего уровня — появление языков с зависимыми типами и специальных средств программирования, облегчающими работу человека с такими программами; появление понятий, методики и деятельности, называемой *type-directed programming* и *type-level programming*; и наконец, заметный рост числа программ, содержащих в своем коде доказательства корректности (по отношению к некоторым утверждениям), выраженных средствами самого языка программирования (а не отдельной надстройки в виде языка математической логики, как в предыдущем примере). Сейчас эта область претерпевает бурное развитие; растет число приложений (практически важный пример: формально корректные *smart contracts*). Можно смело предсказывать, что за следующие десять лет массовые языки (в экологической нише языков C++, Java, C#) также приобретут богатые системы типизации и возникнет достаточно большая практическая область *доказуемо-корректного программирования*, более дорогостоящая, чем традиционное программирование, но окупаемая

[1] https://ru.wikipedia.org/wiki/Соответствие_Карри_—_Ховарда

сторицей там, где стоимость разработки оправдана высокой надежностью критически важных приложений.

4. Метасистемный переход к метавычислениям

Выше приведены примеры массовой человеческой созидательной деятельности, породивших серию метасистемных переходов в программировании. Сама их массовость — это демонстрация действия закона разрастания предпоследнего уровня В. Турчина. Может показаться, что эта эволюция достаточно детерминирована. Но такой вывод ниоткуда не следует. Это «кажимость» задним числом и ретроспективным отбором вех, ведущих к нынешнему состоянию дел.

Концепция В. Турчина ни в коем случае не утверждает, что эволюция детерминирована. Вовсе наоборот, материалом для следующего метасистемного перехода, для построения новой системы управления, могут послужить различные структуры или функции текущего мира. «Причины» начала конкретного метасистемного перехода могут быть самые разнообразные. А когда уже он начался в одной части мира, по сравнению с ним остальные части как бы пребывают в неподвижности, и редко можно увидеть два метасистемных перехода одновременно, происходящих параллельно. Глубокий вопрос (не имеющий у меня ответа): не отнимает ли происходящий МСП чего-то типа «энергии эволюции» (употребляю эти слова чисто метафорически) от развития других частей, так что она уходит в один «канал», как вода прорывает плотину, размывая в одном месте? Или так только кажется?

Однако в человеческом обществе одновременно созидаются много метасистемных переходов, поскольку творцов много и мысли у них разные. Правда, мы часто видим, когда в науке или бизнесе какое-то направление становится на ноги, далее оно может быстро стать популярным из-за того, что реальных творцов мало, а большинство бежит за толпой, веря, что там можно снять сливки, вливая туда новые усилия и финансирования, еще ускоряя процесс. (Тоже разрастание…) А другие направления будут зреть медленно на «голодном пайке». (Это ли не есть иллюзорная «энергия эволюции», о которой вопрос в предыдущем абзаце?)

Именно такая картина сложилась с метавычислениями, независимо задуманными В. Турчиным, А. Ершовым, Й. Футамурой. Идеи и работы Валентина Федоровича я знаю лучше, поэтому буду говорить о нем, не умаляя роль других. В. Турчин целенаправленно размышлял о том, какой метасистемный переход можно и нужно совершить над формальными системами вообще и программами в частности, чтобы продвинуть вперед компьютерные методы и инструменты моделирования мира. Отправная идея простая: сейчас программа является объектом двух метасистем — программиста, ее создающего, и компьютера, ее исполняющего (точнее, сначала компилятора на компьютере, переводящего программу в машинные

коды, а затем ее выполнения — но это уже детали, хотя и такие, которые стоит помнить для последующих аналогий). Далее требуется, чтобы программы стали объектами намного более разнообразных видов деятельности над ними на компьютере, манипулирования для различных целей. А человек останется метасистемой самого верхнего уровня, всё это организующего и управляющего. При этом функции человека станут более творческими, а работа более продуктивной.

Этим В. Турчин задумал весьма крупномасштабный метасистемный переход. Крупные скачки в эволюции совершаются через множество более мелкий шагов, локальных (по пространству и времени) метасистемных переходов. Надо было создать и осуществить первый шаг. Реализующая его система обработки программ была названа *суперкомпилятором* (из-за того, что она, в частности, решала задачу обычной компиляции программ). Здесь В. Турчин следовал типовой схеме метасистемного перехода, хорошо известной в математике по переходу от арифметики к алгебре.[2] А именно, вместо вычислений на конкретных данных по одному пути, детерминированному процессу, суперкомпилятор рассматривает множество исходных данных, промежуточных состояний и путей вычислений, благодаря введенным в их представление переменным[3] (как в алгебре по сравнению с арифметикой) и выполняя определенные операции над частично построенным деревом процессов в символьном виде, в конце концов порождает *остаточную* программу, которая выполняет оставшийся счет более эффективно. В течение 1970-90-х годов В. Турчин развивал методы суперкомпиляции и реализовывал первые суперкомпиляторы (обзор с птичьего полета можно найти в [**7**]). Тогда же его ученики подхватили эти работы и продолжают до сих пор, получая новые научные результаты (все их здесь перечислить невозможно, часть представлена в [**8**]). В других

---

[2] Если бы термин *компьютерная алгебра* (computer algebra) не был уже занят специальной областью обработки математических формул и символьными вычислениями над математическими объектами, то метавычисления можно было бы назвать *алгебраическими вычислениями*. Эта аналогия с арифметикой и алгеброй удачно простирается и сюда: в математике вместо того, чтобы обсчитывать задачу в числах от постановки до решения, сначала задача преобразуют «в общем виде» формулами от описания задачи до того вида, в котором счет более прост. Так и здесь при *специализации* программы методами частичных вычислений или суперкомпиляции она преобразуется «в общем виде» до вида, при котором остаточный счет будет более эффективным по времени или прочим ресурсам.

[3] Где стоят переменные, изначально задает человек, пользователь системы в постановке задачи, а затем в процессе своей работы суперкомпилятор вычисляет сам.

научных школах также развивались и развиваются аналогичные или отличающиеся методы метавычислений (вспомним утверждение о недетерминированности метасистемных переходов).

И что же мы видим через полвека после основания этой отрасли информатики? Теперь мы возвращаемся к основному вопросу этой статьи: где «разрастание предпоследнего уровня» в лице взрыва приложений и создания новых методов разработки софта?

Перед переходом к его обсуждению с выводами «кто виноват?» и «что делать?» полезно сравнить это направление работ В. Турчина и его последователей с реально произошедшими метасистемными переходами, часть которых перечислена выше.

1. Технологии разработки языков и их реализации развивались накоплением частных методов, *ad hoc*, обобщаясь трудами авторов в богатые теории и технологии. Сейчас эта отрасль стала уже «матерой», в которую нелегко войти новичку. Хотя в последнее десятилетие наблюдается ускоренное развитие инструментов создания DSL и реклама этого направления, как решающего определенные проблемы бизнеса, но до конечного состояния, принятого «широкой программистской общественностью» еще далеко. Это разрастание соответствует ожиданиям и прогнозам В. Турчина. Он ценил идею макрогенераторов, как средства автоматизации создания проблемно-ориентированных языков, и сам реализовал систему такого рода под названием *Рефал-макрокод* для ЭВМ Минск-32 [**14**].
2. С другой стороны, интересно, что В. Турчин не рассматривал идею статической типизации как важного метасистемного перехода внутри устройства языков и не придавал такого значения. Да и многие другие этого тогда не понимали, кроме небольшой группы математиков, занимавшихся этой тематикой (Per Martin Löf и др.). Не случайно, созданный Турчиным метаалгоритмический язык Рефал имеет динамическую типизацию (благодаря чему, отметим, Рефал легок в освоении, как и ставшие популярными позднее языки сценариев, scripting languages).

Таким образом, мы можем выделить по крайней мере три крупных метасистемных перехода над программами и языками, развивающимся по своей логике каждый, хотя и с зависимостями и обогащением друг друга:

1. Методы и средства эффективной реализации, оптимизации, распараллеливания (и т.п.) программ, отображения в различные компьютерные архитектуры. Средства и среды разработки программ, учитывающие как удобства программиста, так и целевые архитектуры.

2. Методы и средства верификации программ и доказуемо-корректного программирования, особенно на основе богатых систем типизации, включающих зависимые типы.[4]
3. Методы анализа и преобразования программ, особенно такие глубокие как суперкомпиляция и другие такое же уровня.

Мы называем *метавычислениями*, как правило, лишь третью группу методов. Однако, если посмотреть на общую картину с высоты, то все три являются метадеятельностями над программами, вычислениями над программами, то есть «мета-вычислениями». Говоря о проблемах и подходах к их решению, мы фактически будем иметь в виду широкую область. А решения будем искать на обобщая опыт друг друга.

5. Состояние дел

Длинное введение в тему метавычислений в предыдущих разделах нам понадобилось, чтобы адекватнее понимать стоящие проблемы. В этой версии статьи оставшиеся разделы будут более конспективными. Они основаны на приглашенном докладе автора на ACM Workshop PEPM'22, где не было официальной публикации, а лишь выставлены на сайте файлы тезисов и презентации.[5] Последующий текст является отредактированным переводом тех тезисов.

Я знаю историю российской работы лучше, чем в Европе и США. Что еще более важно, зарубежные публикации не отражают неудачи и приобретенный негативный опыт. Я знаю работу русского научного сообщества изнутри, в том числе, с какими препятствиями мы столкнулись, как мы пытались их преодолеть, чего нам удалось достичь и где мы остановились. Именно поэтому я основываю свой анализ больше на российских работах, чем на работах из других стран.

Основной вывод о причинах недостаточно быстрого движения метавычислений к практике мы видим в том, что существует огромный разрыв между методами метавычисления, с одной стороны, и методами оптимизации в компиляторах, с другой, которые также развивались на протяжении десятилетий и достигли замечательных практических

[4] Здесь нет места для обзора работ по верификации программ, но отметим, что многие современные методы включают в себя в своем виде ту же идею об *обобщенных вычислениях с переменными*, что положила основу суперкомпиляции. Вообще, идеи не ходят по отдельности; они порождаются многократно в разных головах и используются в разных задачах и контекстах, как-то «интерферируя», обобщаясь и выкристаллизовываясь.

[5] https://popl22.sigplan.org/details/pepm-2022-papers/8/Why-are-partial-evaluation-and-supercompilation-still-not-widely-used-in-practice-Re

результатов. Оптимизирующие компиляторы по смыслу своей задачи сконструированы так, что срабатывают «нажатием кнопки», «в черно-ящичном режиме», и пользователь их не замечает. На заре работ по суперкомпиляции была мечта, что будет найдена и реализована совокупность методов, таких что работа суперкомпиляторов будут тоже скрыта от пользователей и (супер)оптимизированные программы будут порождаться в автоматическом режиме. То есть, это будет новый уровень оптимизирующих компиляторов (отсюда и название). Теперь мы понимаем, что эта мечта «нажал кнопку и вот результат» была нереальной и нереализуемой.

Тем не менее, мы по-прежнему уверены, что следующий революционный шаг в разработке программного обеспечения будет основан на метавычислениях, как и думали основатели. На кривой развития технологий Гартнера (Gartner Hype Cycle)[6] мы прошли Пик Завышенных Ожиданий (Peak of Inflated Expectations) и теперь мы находимся на Впадине Разочарования (Trough of Disillusionment), приближаясь к Склону Просветления (Slope of Enlightenment), а до Плато Продуктивности (Plateau of Productivity) еще надо добираться.

Давайте взглянем с высоты птичьего полета на текущее состояние исследований и приложений и подумаем о будущем.

Первая проблема, с которой сталкиваются разработчики алгоритмов анализа и преобразования программ, заключается в том, что это отнимает много компьютерного времени и памяти. Именно это в первую очередь вызывает пессимизм в отношении практических применения. Однако мощность компьютеров экспоненциально растет. Два десятилетия назад мы обнаружили, что прототипы Рефал-суперкомпилятора [**12**] и Java-суперкомпилятора [**6**] начали показывать интересные результаты, когда характеристики персональных компьютеров преодолели символический барьер в 1 ГБ и 1 ГГц. Это произошло на рубеже тысячелетий. С тех пор производительность персональных компьютеров возросла в 100–1000 раз. Тесты Java-суперкомпиляторов, подготовленные два десятилетия назад, теперь работают почти в 100 раз быстрее. А нынешние суперкомпьютеры в 10–100 тысяч раз производительнее, не говоря уже о росте объема памяти.

До недавнего времени казалось, что производительности компьютеров недостаточно для широкого использования даже специализации программ — основной задачи метавычисления. Однако, по аналогии с тем, что бывает с новыми технологиями, можно ожидать, что в какой-то момент времени экспоненциально растущая мощность компьютеров превысит некоторый, заранее не известный, критический уровень, и за короткое время «вдруг» появятся приложения метавычислений. (Например, история быстрого и заранее не ожидавшегося

[6] https://www.gartner.com/en/marketing/research/hype-cycle

появления искусственных нейронных сетей в широкой практике является таким недавним бизнес-кейсом.) Тогда спрос бизнеса привлечет инвестиции в эту сферу, а мы должны быть готовы, имея под рукой зрелые технологии.

Что же делать? Во-первых, чтобы быстрее выйти на этот уровень, мы уже в текущих исследованиях должны использовать всю мощь современных параллельных суперкомпьютеров, облачных вычислений и т.п. Заметим, что считается естественным, когда инженеры, разрабатывающие автомобили и самолеты с использованием систем CAD/CAM, или исследователи в области искусственного интеллекта, машинного обучения и больших данных интенсивно используют суперкомпьютеры. Мы уже привыкли к тестированию и непрерывной интеграции разрабатываемых программных систем, выполняемой на мощных серверах или в облаке. Но по каким-то историческим или психологическим причинам мы не используем такие ресурсы для глубокого анализа и преобразования программ, их оптимизации, распараллеливания, в то время как, например, формальная верификация может в принципе обеспечить такое надежное программное обеспечение, которое невозможно с помощью только тестирования, на что не жалко ресурсов суперкомпьютеров для критических приложений. Из результатов последних десятилетий, возможно, только проверку моделей программ (model checking) можно считать успешным примером, использующим предельную мощность современных (супер)компьютеров. Мы должны следовать ему.

6. Что делать дальше

Как же использовать эту мощность? Мы видим, что наиболее перспективные методы метавычисления не реализуются простыми алгоритмами; они имеют много степеней свободы, точек выбора, которые разрешаются либо быстрыми, но грубыми стратегиями и эвристиками, которые редко дают желаемые результаты, либо методами перебора, требующими больших ресурсов, экспоненциально растущих при увеличении размера задачи. Суперкомпьютеры нужны для них и для алгоритмов между этими крайностями.

Методы метавычислений, которые требуют и могут использовать современную (супер)компьютерную мощность, уже появляются. К примеру, метод насыщения равенствами [**3**], где строится большое пространство кандидатов; многорезультатная компиляция и суперкомпиляция [**9**], при которой вместо принятия конкретных решений рассматривается несколько вариантов остаточной программы и выбирается лучшее по некоторому критерию; поливариантный анализ времен связывания переменных в частичных вычислениях [**5**], в котором генерируется много версий аннотаций.

И последнее, но не менее важное. Система метавычислений должна быть интегрированной человеко-машиной средой, в которой каждая

сторона играет свою роль: система метавычислений выполняет большой объем «технической работы» гарантирует корректность, а человек направляет движение к цели. Решение таких сложных задач — это деятельность взаимодействующих человека и машины. Методы перебора, «грубой силы» (brute force), даже с помощью суперкомпьютеров, не могут решить все проблемы, даже в принципе. Это связано с тем, что машина не знает наших целей, и часто невозможно придумать формальную спецификацию задачи и цели.

Разработка программного обеспечения, как и построение формальных моделей в целом, является человеческой деятельностью, автоматизированной компьютерами. Следует признать, что методы метавычисления перешли границу, за которой роль человека неизбежна. Традиционные оптимизирующие компиляторы остались ниже этой планки.

Поэтому на повестке дня стоит создание интегрированных сред разработки (integrated development environments, IDE), удобных для пользователей, которые включают в себя различные инструменты метавычисления, начиная со специализации программ. К счастью, в последние десятилетия мы наблюдаем большой рост открытых платформ, таких как Eclipse,[7] Visual Studio Code[8] и IntelliJ IDEA,[9] которые вместе с увеличением вычислительной мощности изменили ландшафт программирования по сравнению с тем, что мы имели на рубеже тысячелетий. Мы, разработчики систем метавычислений, опаздываем.

Частичные вычисления (partial evaluation) [**5**] выглядят как первый кандидат для реализации и комфортного использования человеком в интегрированной среде. Их теория довольно хорошо разработана. Этот метод включает первую фазу — анализ времен связывания (binding time analysis), результат которой дает хорошую обратную связь от системы к человеку, ориентировочно показывая, чего можно ожидать от специализации данной программы. Удивительно, но ни один софтверный гигант не инвестировал в разработку таких инструментов. Фреймворк реализации языков Oracle Truffle на платформе GraalVM[10] со встроенным частичным вычислителем является первым перспективным промышленным кейсом. Другие поучительные примеры: язык Julia[11] со специализацией по отношению к типам и фреймворк AnyDSL[12] для разработки предметно-

[7] https://www.eclipse.org

[8] https://code.visualstudio.com

[9] https://www.jetbrains.com/idea

[10] https://docs.oracle.com/en/graalvm/enterprise/20/docs/graalvm-as-a-platform/language-implementation-framework

[11] https://julialang.org

[12] https://anydsl.github.io

ориентированных библиотек с высокой эффективностью благодаря использованию частичных вычислений.

Суперкомпиляция является более сложным методом с еще большим числом степеней свободы, чем частичные вычисления. Роль человека здесь еще важнее. Кроме того, суперкомпиляция и другие технологии, которые строят и анализируют множество вычислительных путей в общих терминах, могут решать и более нетривиальные проблемы, чем специализация. Чем сложнее задачи, решаемые с помощью метавычислений, тем важнее их реализация в комфортной среде для продуктивной работы человека.

Инструменты метавычислений получат широкое признание, когда разработчики программного обеспечения и приложений получат простые в использовании IDE с этими инструментами, которые помогают решать их проблемы, а не усложнять жизнь. Думаю, что это должно быть в центре внимания наших исследований и реализаций, наряду с разработкой новых научных методов.

6. Заключение

Мы рассмотрели истоки идей метавычислений, глубоких методов анализа и преобразования программ в научной философии В. Турчина, давшей общую картину эволюции как последовательности метасистемных переходов. Для В. Турчина эта концепция служила «руководством к действию» по разработке методов суперкомпиляции, как метасистемного перехода над программами, порождающий новый вид обработки программ. Он и другие «отцы-основатели» этой области ожидали быстрого прогресса с практическими приложениями, связанными с высокой автоматизацией разработки программного обеспечения. Но этого не произошло до сих пор.

Тем временем, развитие методов программирования шло по пути совершения других метасистемных переходов. Мы перечислили основные и выделили два наиболее значимых: 1) методы и инструменты создания языков программирования и интегрированных сред разработки программ; 2) методы верификации и богатые системы типизации, включающие зависимые типы. Методы метавычислений уровня суперкомпиляции должны были стать и станут третьим параллельным метасистемным переходом, не менее, а по нашему убеждению и более значимым, чем два упомянутых. Слияние трех линий развития даст еще бóльшее новое качество.

Особенностью метавычислений, как и других методов, стоящих на столь же высоком уровне задач, является то, что не удается (и не удастся) разработать конечную совокупность методов, реализация которых позволит решать достаточно широкие классы практических задач без вмешательства человека (как это случилось с оптимизирующими компиляторами). Все эти методы имеют слишком большое число степеней свободы, требующих

принятия решения для достижения желаемого качества результатов и достижения целей, которые знает человек, но нереально передать машине.

Поэтому на ближайшее будущее эти методы обречены развиваться не как автоматические, «черно-ящичные» средства, а как инструменты для человека, применяемые им сознательно как метасистемой высшего уровня и автоматизирующие его деятельность. Для этого они должны быть погружены в современные среды разработки (IDE) с удобным человеко-машинным интерфейсом. Лет двадцать назад такие системы только начинали свой путь, а сейчас для этого созданы все условия. Дело за разработчиками систем метавычислений. Здесь стоят новые задачи по организации интерфейса с человеком, которые раньше не рассматривались, а теперь давно пора.

Кроме того, в помощь человеку, который не может охватить всё многообразие степеней свободы столь интеллектуальных методов и принятия решений, должны прийти суперкомпьютеры. Использование суперкомпьютеров привычно и естественно в инженерных областях, но почему-то до сих пор не принято, чтобы средства разработки программ работали с суперкомпьютерным счетом на заднем плане. А это необходимо для таких переборных задач как метавычислительные.

С нашей точки зрения, эти два «ключа» — интерактивные инструменты метавычислений, предоставленные человеку в рамках привычных ему сред разработки, и мощность суперкомпьютеров, используемая в процессе разработки программ, — откроет дверь на пути к реализации метасистемного перехода над программами, ожидавшимся В.Ф. Турчиным, А.П. Ершовым и их зарубежными коллегами, и даст качественно новый уровень средств автоматизации разработки эффективных, доказуемо корректных, надежных программ.

Литература

1. Curry H.B., Feys R. Combinatory Logic Vol. I. — Amsterdam : North-Holland, 1958. — https://doi.org/10.2307/3608554
2. Futamura, Y. Partial evaluation of computation process—an approach to a compiler-compiler // Systems Computers Controls. — 1971, Vol. 2, No. 5. — P. 45–50. — https://doi.org/10.1023/A:1010095604496 (reprint).
3. Гречаник С.А. Доказательство свойств функциональных программ методом насыщения равенствами // Программирование. — 2015, № 3. — С. 44–61. — https://elibrary.ru/download/elibrary_23856185_73401772.pdf
4. Howard, W.A. The formulae-as-types notion of construction // To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism. — Boston : Academic Press, 1980. — С. 479–490. — http://www.dcc.fc.up.pt/~acm/howard2.pdf . — (A reprint of an unpublished manuscript from 1969).

5. Jones N.D., Gomard C.K., Sestoft P. Partial Evaluation and Automatic Program Generation. — Prentice-Hall, 1993. — http://www.itu.dk/~sestoft/pebook/pebook.html
6. Klimov And.V. An approach to supercompilation for object-oriented languages: the java supercompiler case study // Proceedings of the First International Workshop on Metacomputation in Russia (July 2–5, 2008, Pereslavl-Zalessky, Russia). — Pereslavl-Zalessky : Ailamazyan University of Pereslavl, 2008. — P. 43–53. — http://meta2008.pereslavl.ru/accepted-papers/meta2008-KlimovA2.pdf
7. Климов Анд.В. О работах Валентина Федоровича Турчина по кибернетике и информатике // Труды SORUCOM-2011. Вторая международная конференция Развитие вычислительной техники и ее программного обеспечения в России и странах бывшего СССР (12–16 сентября 2011, Великий Новгород, Россия). — Великий Новгород : 2011. — С. 149–154. — https://www.computer-museum.ru/histussr/turchin_sorucom_2011.htm
8. Климов Анд.В., Романенко С.А. Краткая история суперкомпиляции в России // Языки программирования и компиляторы — 2017 : труды конференции. — Ростов-на-Дону : Из-во Южного федерального университета, 2017. — С. 147–152. — http://plc.sfedu.ru/files/PLC-2017-proceedings.pdf
9. Klyuchnikov I.G., Romanenko S.A. Multi-result supercompilation as branching growth of the penultimate level in metasystem transition // Perspectives of Systems Informatics. PSI 2011. — Lecture Notes in Computer Science. — Vol. 7162. — P. 210–226. — Berlin, Heidelberg : Springer-Verlag, 2012. —https://doi.org/10.1007/978-3-642-29709-0_19
10. Komorowski J. An introduction to partial deduction // Meta-Programming in Logic. META 1992. — Lecture Notes in Computer Science. — Vol. 649. — Berlin, Heidelberg : Springer, 1992. — P. 49–69. — https://doi.org/10.1007/3-540-56282-6_4
11. Кун Т. Структура научных революций. — М. : Прогресс, 1977. — 300 с. — https://vk.com/doc514662_437573352
12. Немытых А.П. Суперкомпилятор SCP4: общая структура. – М. : URSS, 2007. — 152 с. — ISBN 978-5-382-00365-8. — https://urss.ru/cgi-bin/db.pl?page=Book&id=55806
13. Nordström B., Petersson K., Smith J.M. Programming in Martin-Löf’s type theory: an introduction. — Oxford University Press, 1990. — http://www.cse.chalmers.se/research/group/logic/book/book.pdf
14. Турчин В.Ф. РЕФАЛ-макрокод // Труды Всесоюзного семинара по вопросам макрогенерации. — Тбилиси : ВЦ АН ГССР, 1975. — С. 150–165. — https://pat.keldysh.ru/~roman/doc/Turchin/1975-Turchin--Refal-makrokod.pdf

15. Turchin V.F. The concept of a supercompiler // Transactions on Programming Languages and Systems. — Vol. 8, Iss. 3. — 1986. — P. 292–325. — https://doi.org/10.1145/5956.5957
16. Turchin V.F. Supercompilation: techniques and results // Perspectives of System Informatics. PSI 1996. — Lecture Notes in Computer Science. — Vol. 1181. — Berlin, Heidelberg: Springer, 1996. — P. 227–248. — https://doi.org/10.1007/3-540-62064-8_20
17. Турчин В.Ф. Феномен науки: кибернетический подход к эволюции. — Москва: Наука, 1993. — 295 с. — http://refal.ru/turchin/phenomenon

References

1. Curry H.B., Feys R. Combinatory Logic Vol. I. — Amsterdam : North-Holland, 1958. — https://doi.org/10.2307/3608554
2. Futamura, Y. Partial evaluation of computation process—an approach to a compiler-compiler // Systems Computers Controls. — 1971, Vol. 2, No. 5. — P. 45–50. — https://doi.org/10.1023/A:1010095604496 (reprint).
3. Grechanik S.A. Dokazatelstvo svojstv funkcionalnyx programm metodom nasyshheniya ravenstvami // Programmirovanie. — 2015, № 3. — S. 44–61. — https://elibrary.ru/download/elibrary_23856185_73401772.pdf
4. Howard, W.A. The formulae-as-types notion of construction // To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism. — Boston : Academic Press, 1980. — C. 479–490. — http://www.dcc.fc.up.pt/~acm/howard2.pdf . — (A reprint of an unpublished manuscript from 1969).
5. Jones N.D., Gomard C.K., Sestoft P. Partial Evaluation and Automatic Program Generation. — Prentice-Hall, 1993. — http://www.itu.dk/~sestoft/pebook/pebook.html
6. Klimov And.V. An approach to supercompilation for object-oriented languages: the java supercompiler case study // Proceedings of the First International Workshop on Metacomputation in Russia (July 2–5, 2008, Pereslavl-Zalessky, Russia). — Pereslavl-Zalessky : Ailamazyan University of Pereslavl, 2008. — P. 43–53. — http://meta2008.pereslavl.ru/accepted-papers/meta2008-KlimovA2.pdf
7. Klimov And.V. O rabotax Valentina Fedorovicha Turchina po kibernetike i informatike // Trudy SORUCOM-2011. Vtoraya mezhdunarodnaya konferenciya Razvitie vychislitelnoj texniki i ee programmnogo obespecheniya v Rossii i stranax byvshego SSSR (12–16 sentyabrya 2011, Velikij Novgorod, Rossiya). — Velikij Novgorod : 2011. — S. 149–154. — https://www.computer-museum.ru/histussr/turchin_sorucom_2011.htm
8. Klimov And.V., Romanenko S.A. Kratkaya istoriya superkompilyacii v Rossii // Yazyki programmirovaniya i kompilyatory — 2017 : trudy konferencii. — Rostov-na-Donu : Iz-vo Yuzhnogo federalnogo universiteta, 2017. — S. 147–152. — http://plc.sfedu.ru/files/PLC-2017-proceedings.pdf

9. Klyuchnikov I.G., Romanenko S.A. Multi-result supercompilation as branching growth of the penultimate level in metasystem transition // Perspectives of Systems Informatics. PSI 2011. — Lecture Notes in Computer Science. — Vol. 7162. — P. 210–226. — Berlin, Heidelberg : Springer-Verlag, 2012. —https://doi.org/10.1007/978-3-642-29709-0_19
10. Komorowski J. An introduction to partial deduction // Meta-Programming in Logic. META 1992. — Lecture Notes in Computer Science. — Vol. 649. — Berlin, Heidelberg : Springer, 1992. — P. 49–69. — https://doi.org/10.1007/3-540-56282-6_4
11. Kun T. Struktura nauchnyx revolyucij. — M. : Progress, 1977. — 300 s. — https://vk.com/doc514662_437573352
12. Nemytyx A.P. Superkompilyator SCP4: obshhaya struktura. – M. : URSS, 2007. — 152 s. — ISBN 978-5-382-00365-8. — https://urss.ru/cgi-bin/db.pl?page=Book&id=55806
13. Nordström B., Petersson K., Smith J.M. Programming in Martin-Löf's type theory: an introduction. — Oxford University Press, 1990. — http://www.cse.chalmers.se/research/group/logic/book/book.pdf
14. Turchin V.F. REFAL-makrokod // Trudy Vsesoyuznogo seminara po voprosam makrogeneracii. — Tbilisi : VC AN GSSR, 1975. — S. 150–165. — https://pat.keldysh.ru/~roman/doc/Turchin/1975-Turchin--Refal-makrokod.pdf
15. Turchin V.F. The concept of a supercompiler // Transactions on Programming Languages and Systems. — Vol. 8, Iss. 3. — 1986. — P. 292–325. — https://doi.org/10.1145/5956.5957
16. Turchin V.F. Supercompilation: techniques and results // Perspectives of System Informatics. PSI 1996. — Lecture Notes in Computer Science. — Vol. 1181. — Berlin, Heidelberg: Springer, 1996. — P. 227–248. — https://doi.org/10.1007/3-540-62064-8_20
17. Turchin V.F. Fenomen nauki: kiberneticheskij podxod k evolyucii. — Moskva: Nauka, 1993. — 295 s. — http://refal.ru/turchin/phenomenon