Программные системы и вычислительные методы
Правильная ссылка на статью:
Пекунов В.В. Интерполирующая и экстраполирующая мемоизация в языке Planning C // Программные системы и вычислительные методы. 2025. № 3. DOI: 10.7256/2454-0714.2025.3.37869 EDN: PICMLR URL: https://nbpublish.com/library_read_article.php?id=37869

Интерполирующая и экстраполирующая мемоизация в языке Planning C

Пекунов Владимир Викторович

доктор технических наук

Инженер-программист, ОАО "Информатика"

153000, Россия, Ивановская область, г. Иваново, ул. Ташкентская, 90

pekunov@mail.ru

Статья из рубрики "Математическое моделирование и вычислительный эксперимент"

DOI:

10.7256/2454-0714.2025.3.37869

EDN:

PICMLR

Дата направления статьи в редакцию:

12-04-2022

Аннотация: В данной работе рассматриваются возможности повышения скорости исполнения программ, реализующих преимущественно математические алгоритмы, с помощью некоторых специальных видов мемоизации. Проведен краткий обзор существующих основных подходов к мемоизации, сделан вывод о недостаточной изученности возможностей явной программной мемоизации, основанной на том или ином способе приближения отсутствующих в кэше мемоизации результатов. Анализируются возможности такой мемоизации, описываются синтаксис и семантика возможных программных конструкций, указывающих на необходимость ее включения для функций/процедур (void-функций). Проводится апробация предлагаемых вариантов мемоизации, показано, что для некоторых математических алгоритмов возможно существенное ускорение работы при достаточно невысокой погрешности. Новизна данного исследования состоит в том, что впервые предложены и описаны синтаксис, семантика и основные механизмы реализации явной программной мемоизации, основанной на интерполяции (нейронными сетями прямого распространения или методом группового учета аргументов) или линейной экстраполяции. Данная мемоизация

вводится в язык Planning C. Сформулированы условия оправданности мемоизации. Для предложенного варианта мемоизации вводится понятие группирующего параметра, позволяющего использовать наборы интерполяторов для различных комбинаций входных аргументов мемоизированной процедуры/функции с целью снижения дополнительных временных затрат на обучение интерполятора и повышение правдоподобности его результатов. Также введено понятие порядкового параметра, используемого для установления порядка ключевых точек экстраполирующей мемоизации. Адекватность предложенных подходов и алгоритмов мемоизации показана на ряде примеров из области численного моделирования.

Ключевые слова:

мемоизация, предикция, интерполятор, экстраполятор, синтаксические конструкции, язык программирования, апробация, параллельные вычисления, нейронная сеть, МГУА

Введение

Наиболее часто под мемоизацией понимают кэширование результатов исполнения функций, преимущественно в функциональных языках [1]. Однако на практике мемоизация может трактоваться более широко, как прием, позволяющий предсказывать результат выполнения некоторого программного блока (функции, цикла, произвольного фрагмента кода, произвольной серии инструкций процессора) на основании известных результатов предыдущих сеансов исполнения этого блока. Это достаточно мощный прием, позволяющий в ряде случаев значительно сократить время выполнения программы за счет исключения повторного вычисления ее отдельных блоков. Рассмотрим данную проблему более подробно.

Мемоизацию, по типу реализации можно разделить на аппаратную и программную. При этом на аппаратном уровне она может быть реализована либо на уровне инструкций процессора, либо на уровне аппаратной поддержки кэшей результатов функций и/или циклов. Программная реализация может быть либо встроена в транслятор (который автоматически определяет потенциально безопасные функции, которые могут быть эффективно мемоизированы), либо представлена на уровне специальных инструкций, вставляемых в код программистом.

По наличию аппроксимации мемоизацию можно разделить на строгую (без аппроксимации) и нестрогую (интерполирующую или экстраполирующую). Нестрогая мемоизация предполагает наличие попыток выдать приближенный (интерполированный или экстраполированный) результат для набора аргументов, которого нет в собранном кэше таких наборов, тогда как строгая мемоизация ограничивается выдачей результатов только для кэшированных наборов аргументов. При этом очевидно, что нестрогая мемоизация способна давать существенно более значительное сокращение общего времени выполнения программы, разумеется, если она находится под строгим контролем, не пропускающим ирреальные прогнозы результатов (по этой причине более предпочтительной является именно программная мемоизация).

Рассмотрим некоторые стандартные языки программирования и исследовательские работы, в которых представлены различные варианты мемоизации.

В языке Haskell мемоизация на уровне языка не поддерживается, используются явные синтаксические конструкции. Язык позволяет для некоторой комбинации аргументов

определить прием, позволяющий строго мемоизировать любую функцию с такими аргументами. Это не вполне универсальный прием, хотя потенциально вполне позволяет реализовать даже интерполирующую мемоизацию. В языке Common Lisp [2] строгая мемоизация встроена на уровне транслятора. В языке Perl [3] мемоизация в язык не встроена, но реализована в виде библиотеки, позволяющей применять соответствующие явные синтаксические конструкции (строгая мемоизация).

В работе [4] описана автоматическая программная мемоизация, реализуемая путем перехвата вызовов к динамически подгружаемым функциям на уровне исполняемого кода. Таким образом, реализуется универсальный подход к строгой мемоизации, независимо от используемого языка программирования, к сожалению также неконтролируемой.

В работе [5] мемоизация результатов функций и циклов реализуется на уровне специального процессора, без интерполяции/экстраполяции результатов. При обращении к функции сначала порождаются два спекулятивных потока, в одном из которых предполагается, что можно воспользоваться результатом мемоизации (в этом потоке исполнение функции пропускается), во втором – нельзя (в этом потоке функция начинает исполняться). Одновременно с этим идет проверка на возможность использования результата мемоизации и, в зависимости от решения, принимается результат первого или второго потока.

В работе [6] представлена мемоизация с применением аппаратных кэшей данных, явно доступных программе.

В работах [7],[8],[9] описывается условный вариант нестрогой мемоизации как спекулятивных вычислений на уровне процессора. При этом используется лишь линейная целочисленная предикция (чтобы не вносить больших дополнительных накладных расходов, возможных, например, при работе с вещественными числами): по последнему значению [7], с шагом [8], контекстная [9], или гибридная [8]. Заметим, что прогностические стратегии используются процессором также в технологии предсказания ветвлений [10, 11] (в сложных случаях даже используется прогнозирование с применением нейронной сети [11], в частности линейного персептрона), что повышает эффективность работы современных процессоров с конвейеризацией вычислений. Неким вариантом нестрогой мемоизации можно считать и прогнозирование значений переменных, например, на уровне транзакционной памяти, однако в соответствующих работах (см., например, [12]) речь обычно идет о прогнозировании целочисленных значений отдельных ячеек памяти или переменных, а не об интерполяции/экстраполяции вещественных значений.

Весьма интересной представляется работа [13], описывающая вариант неявной нестрогой программной мемоизации на базе нейросетевой аппроксимации. Компилятор обнаруживает фрагменты кода, результаты которых аппроксимируемы нейронной сетью и заменяет их вызовами нейросетевого модуля. Отметим, что это достаточно универсальный прием, но сложности с автоматическим поиском вышеуказанных фрагментов кода и потенциально невысокое качество такой предикции для некоторых из данных фрагментов существенно снижает его ценность.

Резюмируя данный краткий обзор, сведем его результаты в таблицу 1.

Таблица 1. Результаты обзора известных вариантов реализации мемоизации

Мемоизация	Строгая	Нестрогая
Аппаратная на уровне инструкций	[5]	[7],[8],[9],[12]
Аппаратная на уровне доступного программе специального кэша	[6]	
Программная, неявная (встроена в компилятор)	Common Lisp [2], также универсальный подход [4]	[13]
Программная, явная (на уровне специальных инструкций)	Perl [3], также Haskell	

Достаточно очевидно, что именно нестрогая явная программная мемоизация, отмеченная выше как наиболее перспективная, в настоящее время практически не исследована (по крайней мере, найти работы по данной проблематике автору не удалось). При этом весьма интересным представляется рассмотрение именно варианта с нецелочисленной предикцией (поскольку наибольший эффект мемоизация дает в случае кэширования/прогнозирования результатов выполнения трудоемких блоков с хорошо определяемыми зависимостями «вход-выход», что в наибольшей степени характерно для математических алгоритмов, обычно имеющих дело с вещественными числами), который также недостаточно исследован в смысле применения мемоизации. Поэтому данная тематика представляется достаточно актуальной.

Целью данной работы является исследование возможности повышения скорости выполнения программ на языках высокого уровня с помощью нестрогой явной программной мемоизации. Для достижения данной цели поставим следующие задачи: а) сформулировать понятие интерполирующей/экстраполирующей мемоизации; б) предложить синтаксис и семантику специальных директив включения такой мемоизации для процедур/функций, которые могут быть интегрированы, например, в язык Planning C (расширение C++, [14]) его штатными средствами; в) провести апробацию некоторых вариантов предлагаемой мемоизации для процедур/функций, решающих типовые подзадачи численного моделирования с потенциально хорошо интерполируемыми или экстраполируемыми результатами.

Интерполирующая/экстраполирующая мемоизация

Пусть речь идет о мемоизированной процедуре/функции, выполняющей некий алгоритм, результаты которого описываются достаточно гладкой, но неизвестной заранее математической функцией. Такие результаты, теоретически, могут быть вычислимы (хотя бы в некоторых точках) с помощью интерполяции или экстраполяции, если известен набор ключевых точек. Заметим, что

а) набор ключевых точек (параметры функции, результат) может быть получен как побочный эффект мемоизации;

б) при определенных условиях вполне возможна *интерполирующая мемоизация*, когда для текущих параметров сохраненного результата нет, но он может быть достаточно легко вычислен;

в) в некоторых случаях возможна и *экстраполирующая мемоизация*, когда последовательность результатов функции является неким временным рядом, имеющим закономерности, и очередное значение функции может быть предсказано неким сравнительно низкозатратным алгоритмом.

Следует подчеркнуть, что интерполирующая/экстраполирующая мемоизация такого рода будет давать *положительный эффект только при соблюдении следующих условий*:

а) время вычисления интерполированного/экстраполированного результата C строго меньше среднего времени выполнения процедуры/функции T;

б) имеет место неравенство P + K×C < K×T, где P – время обучения предиктора, K – некоторое количество итераций, на которых обученный предиктор дает приемлемые по точности результаты;

в) выполняется периодический контроль качества предиктора, когда для заданного набора входных параметров вычисляется прогноз V, затем запускается мемоизированная процедура/функция, вычисляющая верный результат B, и, наконец, сравниваются V и B. Если погрешность прогноза не превышает некоторой заданной точности, то предиктор валиден и будет использоваться еще некоторую серию вызовов. В противном случае предиктор подлежит переобучению по имеющемуся набору мемоизированных точек, возможно, в сочетании с дополнительным набором новых точек, получаемых в результате прямого вызова процедуры/функции.

Такая мемоизация может быть применена, например, если процедура/функция имеет достаточно высокую вычислительную трудоемкость, например, решает прямую задачу химической кинетики, или интегрирует дифференциальное уравнение, результат расчета которого входит в какой-либо еще циклический математический алгоритм. Следует заметить, что, если рабочим предиктором является, например, нейронная сеть, линейный экстраполятор или полином, построенный методом группового учета аргументов (МГУА [15]), то возможен любопытный побочный эффект: мемоизированная процедура/функция может, фактически, стать интерфейсом обучения и вычисления отклика такого элемента – тем самым *нейронные сети, линейные экстраполяторы и МГУА-полиномы неявно вводятся в язык программирования* и могут быть применены в нем для самых различных программных целей.

Синтаксис и семантика директив включения мемоизации

Включение режима мемоизации для процедуры/функции производится с помощью специальной директивы, вставляемой в отдельной строке непосредственно перед заголовком:

директива_мемоизации = «**#pragma**» пробелы «**memoization**» [пробелы] «**(**» [пробелы] карта_параметров [пробелы] «**)**» [[пробелы] (МГУА | нейросеть|экстраполятор) [вид_контроля]]

карта_параметров = параметр {[пробелы] «**,**» [пробелы] параметр}

параметр = {«*****»} вид_параметра

вид_параметра = входной | выходной | группирующий | порядковый

входной = «**i**»

выходной = «**o**»

группирующий = «**g**»

порядковый = «**t**»

МГУА = «**mgua**» [пробелы] «**(**» точность «**)**»

точность = вещественное_число_от_0_до_1

нейросеть = «**feed_forward**» пробелы «**(**» точность «**,**» коэффициент_обучения «**,**» макс_число_итераций «**,**» [описатель_нелинейных_слоев] «**)**»

экстраполятор = «**lin_extrapolator**» [пробелы] «**(**» точность «**,**» порядок_экстраполятора «**)**»

коэффициент обучения = вещественное_число

порядок_экстраполятора = целое_число_большее_нуля

описатель_нелинейных_слоев = число_нейронов_в_слое «**,**» актив_функция [«**,**» число_нейронов_в_слое «**,**» актив_функция]

актив_функция = экспон_сигмоида | линейная | гиперб_тангенс | ReLU

экспон_сигмоида = «**e**»

линейная = «**l**»

гиперб_тангенс = «**h**»

ReLU = «**r**»

вид_контроля = автоматический | по_условию

автоматический = «**controlled**» «**(**» число_разгонных_точек «**,**» число_предицируемых_точек «**)**»

по_условию = «**conditions**» «**(**» условие_отбора_точки_в_историю «,» условие_предикции «**)**»

условие_отбора_точки_в_историю = логическое_выражение

условие_предикции = логическое_выражение

Дадим некоторые пояснения. *Карта_параметров* указывается всегда, ее элементы специфицируют вид параметра – входной, выходной, порядковый или группирующий. Если параметр представляет собой указатель на входное/выходное/группирующее значение, то перед символом вида указывается оператор разыменования «*****». Если мемоизация производится для функции, то ее *результат всегда является одним из выходных параметров, заданным неявно*.

Группирующий параметр введен для упрощения вычисления предикторов и повышения их качества. Если он определен (в таком случае он должен быть единственным!), то для каждого из обнаруженных в ходе работы значений этого параметра генерируется отдельный предиктор, обучаемый только по точкам, полученным при передаче в процедуру/функцию данного значения группового параметра. В частности, такой режим очень удобен при мемоизации процедур/функций, вычисляющих неким математическим алгоритмом значения некоторого поля. В этом случае попытки обучения единого

предиктора для всех узловых значений поля могут быть малоуспешными или занимать чрезмерно большое время на обучение, в то время как обучение отдельного предиктора для каждого узла способно дать заметно лучшие результаты. В этом случае следует ввести мемоизируемую процедуру, обрабатывающую за один раз один узел, принимающую ряд входных и один выходной (узловое значение поля) параметры, а также группирующий параметр – номер узла, все остальное среда исполнения проделает автоматически.

Порядковый параметр имеет значение только для экстраполирующей мемоизации. В этом случае его значение используется для установления порядка, в котором будут рассматриваться значения выходного параметра при построении линейного экстраполятора. Порядковый параметр может быть только один, кроме того, он должен быть первым параметром мемоизируемой процедуры/функции. При этом наличие прочих входных параметров возможно, но они не будут приниматься во внимание. Выходной параметр должен быть только одним (явно указывается в карте параметров или под ним подразумевается результат функции, если мемоизируется функция).

Режим обычной мемоизации включается, если в вышеуказанной директиве не указано ни утверждения «**mgua**», ни утверждения «**feed_forward**», ни утверждения «**lin_extrapolator**».

Режим интерполирующей/экстраполирующей мемоизации без контроля включается, если в директиве указано утверждение «**mgua**» или утверждение «**feed_forward**» или утверждение «**lin_extrapolator**», но не содержится указание «**controlled**» или «**conditions**». В данном режиме по умолчаниию выполняется обычная мемоизация, а для *явного* вызова предиктора используется специальный синтаксис с префиксом

«**predict_**» идентификатор_процедуры «**(**» параметры_вызова «**)**»

Для режима МГУА в директиве указывается только требуемая точность, для линейной экстраполяции – требуемая точность и число коэффициентов экстраполятора, для нейросети – требуемая точность, параметр для метода обратного распространения ошибки и максимальное количество итераций данного метода. Если *описатель нелинейных слоев отсутствует*, то нейронная сеть представляет собой однослойный линейный персептрон (число нейронов равняется количеству вычисляемых предиктором элементов). Если же *описатель присутствует*, то нейронная сеть имеет указанные нелинейные слои с указанными активационными функциями, а также линейный выходной слой (число нейронов в котором равняется количеству вычисляемых предиктором элементов).

Режим интерполирующей мемоизации с автоматическим контролем включается при наличии указания «**controlled**». При этом процедура/функция начинает работу в режиме обычной мемоизации, собирая историю ключевых точек на протяжении величины **число_разгонных_точек**. Вводится внутренний параметр – коэффициент доверия результатам **w**, изначально равный единице. Далее включается режим *контролируемой предикции*, в котором: а) выполняются вызовы предиктора с возвратом спрогнозированных результатов (на протяжении числа вызовов, равного величине **w*число_предицируемых_точек**), б) осуществляется контроль (вычисляется предиктор, затем для тех же параметров вызывается исходная процедура и результаты сравниваются) – если контроль по точности проходит, то **w** увеличивается и переходим к (а), и так далее. Если же контроль по точности не проходит, то **w** уменьшается, предиктор пересчитывается и переходим к (б).

Режим интерполирующей мемоизации с контролем по условию включается при наличии указания «**conditions**». Если при очередном обращении к процедуре/функции выполняется **условие_предикции**, то выполняется предикция с возвратом соответствующих результатов. Если же это условие не выполняется, то для получения результатов вызывается исходная процедура/функция. Если при этом выполняется **условие_отбора_точки_в_историю**, то текущая пара (параметры, результат) включается в историю. Как только размер собранной истории превысит максимальный (который определяется при вызове функции **set_memoization_max_history**), предиктор автоматически переобучается.

Приведем примеры применения директивы для классической мемоизации функции вычисления числа Фибоначчи:

```
#pragma memoization(i)

double fibb(int a) {

if (a < 3) return 1;

else return fibb(a-1)+fibb(a-2);

}

#pragma memoization(i,o)

void fibb(int a, double &result) {

if (a < 3) result = 1;

else {

double b, c;

fibb(a-1, b);

fibb(a-2, c);

result = b+c;

}

}
```

Апробация

1. Пусть решается следующая математическая задача: дана труба с неизвестным коэффициентом теплоотдачи стенок. По трубе движется жидкость (вода), ее температура на входе и на выходе известна, также известна температура окружающей среды. Необходимо определить коэффициент теплоотдачи стенок.

Данная задача может решаться как проблема нелинейной оптимизации. Минимизируется функция ошибки расчета F(alpha), зависящая от коэффициента теплоотдачи alpha. Для определения ошибки расчета задается известная температура на входе трубы и текущее значение коэффициента теплоотдачи alpha, затем, в численном эксперименте определяется температура на выходе $T_{эксп}$, которая сравнивается с известной температурой $T_{вых}$. То есть, проблема имеет вид:

$$alpha_{рез} = arg\ min(F(alpha));$$

$$F(alpha) = (T_{эксп}(alpha) - T_{вых})^2.$$

Итак, пусть $T_{эксп}$ находится путем численного интегрирования уравнения теплопроводности до установления при заданном alpha. Пусть расчетная сетка достаточно велика (имеет несколько тысяч узлов). Поместим этот расчет в процедуру и применим интерполирующую мемоизацию. Необходимые условия применимости интерполирующей мемоизации будут выполнены, если использовать нейросетевой интерполятор небольшого размера – тогда время его обучения и вычисления будет меньше, чем время численного решения уравнения теплопроводности, причем погрешность может быть невелика, поскольку $T_{эксп}(alpha)$, скорее всего, является достаточно гладкой функцией. Воспользуемся трехслойной нейронной сетью с пятью нейронами в первом слое (гиперболический тангенс), тремя – во втором (гиперболический тангенс) и одним линейным нейроном в третьем слое. Применим автоматический контроль погрешности. Директива мемоизации будет иметь вид:

```
#pragma memoization(i,o) feed_forward(0.001, 0.05, 2000, 5, h, 3, h) controlled(7,2)

void get_t(double alpha, double & t1) {

// Интегрирование уравнения теплопроводности (при заданном alpha) до установления, вычисляется температура на выходе t1

}
```

Как показали результаты вычислений и замеров, данная программа нашла значение alpha за 1144,57 с. При этом аналогичная программа без мемоизации нашла значение alpha за 10163 с. Погрешность варианта с мемоизацией составила менее 0,01%. Таким образом, при очень малой погрешности, для данной задачи интерполирующая мемоизация позволила получить результат в 8,88 раза быстрее. Обычная мемоизация не смогла в данном случае дать существенного выигрыша (что также было подтверждено экспериментально – время решения существенно превысило результат интерполирующей мемоизации), поскольку входные значения alpha являются вещественными числами и практически не повторяются в процессе решения.

2. Пусть решается двумерная аэродинамическая задача, осложненная химической кинетикой (модель для такого случая взята из работы [16]): моделируется обдувание здания, на крыше которого происходит активное выделение и горение метана, образуются продукты горения. Наиболее вычислительно затратной процедурой здесь является интегрирование подсистемы уравнений химической кинетики (15 уравнений) в каждом узле расчетной области. Выделим данную процедуру в отдельный блок и попробуем применить интерполирующую мемоизацию.

Прежде всего, заметим, что вряд ли возможно применение единого интерполятора для всех узлов расчетной области, поскольку подсистема уравнений химической кинетики достаточно непроста и вряд ли имеет тривиальное аналитическое решение, которое может быть эффективно интерполировано нейронной сетью небольшой сложности. Поэтому, именно в данном случае представляется оправданным применение группирующего параметра – номера узла, который позволил бы обучить для каждого узла свой локальный интерполятор. Попытка применить в данном случае нейронную сеть даже в таком качестве завершилось неуспехом – мемоизация не только не дала

выигрыша по времени, но даже замедлила решение. Поэтому было принято решение попытаться применить более быстро вычисляемый МГУА с автоматическим контролем погрешности мемоизации. Директива мемоизации имеет вид:

```
#pragma memoization(g,i,i,i,o) mgua(0.0000001) controlled(800,10)

void get_kinetics(int group, double _time, double Tk, float INC[15], float OUTC[15]) {

// Интегрирование уравнений химической кинетики с начальными концентрациями INC, вычисляются конечные концентрации OUTC

}
```

Здесь **group** – группирующий параметр (номер узла), **_time** – текущее время моделирования, **Tk** – температура в текущем узле, **INC** – массив входных концентраций веществ, **OUTC** – массив выходных концентраций веществ.

Как показали результаты вычислений и замеров, моделирование на интервале модельного времени в 0,485 секунды при использовании мемоизации заняло 205,6 секунд, а при ее отсутствии – 271,91 секунды. При этом относительная погрешность не превысила 15-16%. С учетом достаточно высокой сложности задачи такой результат может быть признан вполне удовлетворительным.

3. Рассмотрим предыдущую задачу, но применим мемоизацию не для процедуры А интегрирования уравнений химической кинетики, а для процедуры Б, оценивающей время исполнения процедуры А во всех узлах. Это необходимо при параллельном решении предыдущей задачи с применением геометрического параллелизма, когда расчетная область поблочно распределяется по процессорам. При этом в каждом таком блоке присутствуют узлы, время интегрирования уравнений химической кинетики в которых существенно различается. Для более эффективного распараллеливания в таком случае, используются оценки времени интегрирования, которые позволяют оперативно перераспределить узлы по процессорам с целью максимального выравнивания их загрузки. Такие оценки для каждого конкретного узла часто представляют собой временной ряд с периодически наблюдающимися локальными закономерностями, обусловленными, например, характером изменения температуры протекания реакций в узле. Соответственно, можно попытаться применить экстраполирующую мемоизацию с порядковым параметром **i** и выходным массивом прогнозов времени в узлах **stat**. Директива мемоизации имеет вид:

```
#pragma memoization(t,o) lin_extrapolator(eps, 2) controlled(50, 1)

void prognose(int i, double stat[NY][NX]) {

// Заполнение массива stat замерами времени, потраченного на интегрирование уравнений химической кинетики во всех узлах. Таким образом, при отсутствии экстраполяции считаем, что данные замеры являются прогнозом времени интегрирования для следующей итерации

}
```

Такая мемоизация показала достаточно хорошие результаты при порядке экстраполятора, меньшем трех (при более высоких порядках экстраполятор был склонен выдавать достаточно нереальные прогнозы). При этом время решения задачи (на четырех процессорах) без экстраполяции/балансировки составило 168,88 с, при

экстраполяции/балансировке с первым порядком экстраполятора – 73,19 с, со вторым порядком экстраполятора – 70,05 с. Время решения задачи в непараллельном варианте составляло 263,71 с, то есть в данном случае распараллеливание было оправданным, а экстраполирующая мемоизация – достаточно эффективной. Поскольку в данном случае прогнозирование применялось не для переменных задачи, а для оценок времени их расчета, вносимая погрешность здесь отсутствует.

Выводы

Итак, в данной работе проведен обзор существующих основных подходов к мемоизации, сформулировано понятие явной интерполирующей/экстраполирующей программной мемоизации, кратко описаны механизмы ее работы, описаны синтаксис и семантика директив включения такой мемоизации для функций/процедур (void-функций) языка Planning C. Проведена апробация предложенного программного формализма для некоторых подзадач численного моделирования и связанных с ними подзадач прогнозирования. Показано существенное ускорение работы для таких задач, причем при адекватном контроле погрешности дополнительная вносимая ошибка составила от 0,01% до 15-16%. При этом максимальные значения 15-16% наблюдалась только в наиболее сложном для прогнозирования случае -- при интерполирующей мемоизации результатов решения системы уравнений химической кинетики.

Библиография

1. Городняя Л.В. Парадигмы программирования. Часть 4. Параллельное программирование //Новосибирск. Препринт ИСИ СО РАН. – URL: http://www.iis.nsk.su/files/preprints/gorodnyaya_175.pdf (дата обращения: 05.11.2021).
2. Mayfield, James, Timothy W. Finin and M. R. Hall. Using automatic memoization as a software engineering tool in real-world AI systems // Proceedings the 11th Conference on Artificial Intelligence for Applications (1995): pp.87-93.
3. Perl 5.34.0 Documentation. URL: https://perldoc.perl.org/Memoize (дата обращения: 05.11.2021)
4. Arjun Suresh, Bharath Narasimha Swamy, Erven Rohou, and Andre Seznec. Intercepting functions for memoization: A case study using transcendental functions // ACM Trans. Architec. Code Optim. 12, 2, Article 18 (June 2015), 23 pages. DOI: http://dx.doi.org/10.1145/2751559
5. Y. Kamiya, T. Tsumura, H. Matsuo and Y. Nakashima. A Speculative Technique for Auto-Memoization Processor with Multithreading // Proc. International Conference on Parallel and Distributed Computing, Applications and Technologies, 2009, pp. 160-166, DOI: 10.1109/PDCAT.2009.67.
6. G. Zhang and D. Sanchez. Leveraging Hardware Caches for Memoization // IEEE Computer Architecture Letters, vol. 17, no. 1, pp. 59-63, 1 Jan.-June 2018, DOI: 10.1109/LCA.2017.2762308.
7. M. H. Lipasti and J. P. Shen. Exceeding the dataflow limit via value prediction // Proceedings of the 29th Annual IEEE/ACM International Symposium on Microarchitecture. MICRO 29, 1996, pp. 226-237, doi: 10.1109/MICRO.1996.566464.
8. B. Calder, G. Reinman and D. M. Tullsen. Selective value prediction // Proceedings of the 26th International Symposium on Computer Architecture (Cat. No.99CB36367), 1999, pp. 64-74, doi: 10.1109/ISCA.1999.765940.
9. Yiannakis Sazeides and James E. Smith. The predictability of data values // In Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture (MICRO 30). IEEE Computer Society, USA,1997, pp. 248-258.
10. Evers, M., Yeh, T.Y. Understanding branches and designing branch predictors for high

performance microprocessors // Proceedings of the IEEE 89, 1610-1620 (2001).
11. Monchiero M., Palermo G. (2005) The Combined Perceptron Branch Predictor. // In: Cunha J.C., Medeiros P.D. (eds) Euro-Par 2005 Parallel Processing. Euro-Par 2005. Lecture Notes in Computer Science, vol 3648. Springer, Berlin, Heidelberg. DOI:https://doi.org/10.1007/11549468_56
12. Salil Pant and Greg Byrd. A case for using value prediction to improve performance of transactional memory // In TRANSACT '09: 4th Workshop on Transactional Computing, feb 2009. URL: http://transact09.cs.washington.edu/35_paper.pdf (дата обращения: 31.08.2020).
13. Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. Neural acceleration for generalpurpose approximate programs // In Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-45). 449-460. DOI:http://dx.doi.org/10.1109/MICRO.2012.48.
14. Пекунов В.В. Язык программирования Planning C. Инструментальные средства. Новые подходы к обучению нейронных сетей. – LAP LAMBERT Academic Publishing, 2017. – 171 с.
15. Дюк, В., Самойленко, А. Data mining: учебный курс. СПб: Питер, 2001.
16. Пекунов, В.В. Новые методы параллельного моделирования распространения загрязнений в окрестности промышленных и муниципальных объектов // Дис. докт. тех. наук. – Иваново, 2009. – 274 с.

Результаты процедуры рецензирования статьи

В связи с политикой двойного слепого рецензирования личность рецензента не раскрывается.
Со списком рецензентов издательства можно ознакомиться здесь.

Рецензируемая статья посвящена применению нестрогой явной программной интерполирующей и экстраполирующей мемоизации на языке программирования Planning C для минимизации затрат машинного времени на выполнение компьютерных программ.
Методология исследования базируется на обобщении литературных источников по теме работы, апробации предложенных авторами синтаксиса и семантики специальных директив включения явной программной интерполирующей и экстраполирующей мемоизации для процедур/функций в языке Planning C.
Актуальность исследования обусловлена необходимостью сокращения времени выполнения компьютерных программ за счет исключения повторного вычисления ее отдельных блоков.
Научная новизна представленного исследования, по мнению рецензента, заключается в авторской формулировке понятия явной интерполирующей/экстраполирующей программной мемоизации, апробации предложенного программного формализма для некоторых подзадач численного моделирования и связанных с ними подзадач прогнозирования.
В статье структурно выделены следующие разделы: Интерполирующая/экстраполирующая мемоизация, Синтаксис и семантика директив включения мемоизации, Апробация, Выводы, Библиография.
Авторами стати рассмотрены различные варианты мемоизации в некоторых стандартных языках программирования: Haskell, Common Lisp, Perl. Результаты обзора известных вариантов реализации мемоизации представлены в отдельной таблице, наглядно иллюстрирующей освещение в рассматриваемых публикациях строгой и нестрогой мемоизации с применением аппаратной мемоизации на уровне инструкций и на уровне доступного программе специального кэша, а также программной неявной (встроенной в

компилятор) и программной явной (на уровне специальных инструкций) мемоизации.
В статье рассмотрены условия, при которых возможны интерполирующая и экстраполирующая мемоизация могут дать положительный эффект; синтаксис и семантика директив включения мемоизации с необходимыми пояснениями; апробация внесенных предложений на примере решения трех конкретных математических задач.
Выводы по статье содержат отражение основных результатов проведенного исследования.
Библиографический список включает 14 наименований источников, представленных научными статьями отечественных и зарубежных авторов, на которые в тексте приведены адресные ссылки, свидетельствующие о наличии в публикации апелляции к оппонентам.
По рецензируемому материалу можно высказать некоторые замечания и пожелания.
Во-первых, в тексте не выделены такие общепринятые в современных журнальных научных публикациях разделы как Введение, Материалы и методы исследования.
Во-вторых, представляется, что заключительная фраза в статье: «дополнительная вносимая ошибка составила от 0,01% до 15-16% (в зависимости от конкретной задачи)», требует более детального пояснения, поскольку ошибка 15% – весьма существенна.
В-третьих, описание библиографических источников, заключенных в квадратные скобки, целесообразно перенести из основного текста статьи в библиографический раздел, сопроводив их соответствующими ссылками.
Материал соответствует направлению журнала «Кибернетика и программирование», содержит элементы научной новизны и практической значимости, рекомендуется к опубликованию после доработки в соответствии с высказанными замечаниями.

Результаты процедуры повторного рецензирования статьи

В связи с политикой двойного слепого рецензирования личность рецензента не раскрывается.
Со списком рецензентов издательства можно ознакомиться здесь.

Статья посвящена исследованию подходов к мемоизации – техники оптимизации вычислений, позволяющей ускорить выполнение программ за счет хранения и повторного использования ранее вычисленных результатов. Основное внимание уделено разработке методов интерполирующей и экстраполирующей мемоизации, которые могут применяться для задач численного моделирования и других вычислительно сложных операций.
Автор использует комплексный подход к изучению мемоизации, включающий обзор существующих методов, разработку собственных синтаксических конструкций для языка Planning C, а также экспериментальное тестирование предложенных методов на примерах численных задач. Приведенная апробация включает сравнение производительности с применением и без применения мемоизации.
Тема исследования является актуальной в контексте растущей потребности в оптимизации вычислительных ресурсов, особенно при решении задач численного моделирования. Применение мемоизации позволяет значительно ускорить вычисления в задачах, связанных с прогнозированием, численной интеграцией и оптимизацией.
Статья предлагает оригинальный подход к реализации интерполирующей и экстраполирующей мемоизации в языке программирования Planning C. Научная новизна заключается в разработке специального синтаксиса и семантики директив мемоизации, что позволяет легко интегрировать этот механизм в программный код. Экспериментально подтверждена эффективность предложенного подхода.
Текст статьи логично структурирован: от теоретического введения и обзора

существующих методов до описания разработок и их апробации. Научный стиль изложения соответствует требованиям академических публикаций. Автор четко формулирует цели и задачи, обосновывает выбор методов и приводит результаты в табличной и текстовой форме. Библиографический список содержит актуальные и релевантные источники.

Выводы статьи подтверждают значительный потенциал предложенного подхода к мемоизации для ускорения вычислений. Автору удалось показать, что разработанные методы эффективны как в численном моделировании, так и в задачах прогнозирования.

Статья представляет собой значительное научное исследование, но имеются направления, которые могут дополнительно усилить ее вклад в область программирования и численного моделирования.

Во-первых, автору рекомендуется рассмотреть возможность применения предложенных методов мемоизации к более широкому спектру задач. Например, расширить исследования на задачи обработки данных в реальном времени или задачи, связанные с оптимизацией в машинном обучении. Это позволит продемонстрировать универсальность подхода и его применимость за пределами численного моделирования.

Во-вторых, перспективным направлением развития является создание инструментов для автоматической интеграции предложенных директив мемоизации в существующий программный код. Такой инструмент мог бы анализировать исходный код на предмет идентификации подходящих функций для мемоизации и автоматически предлагать оптимальные параметры. Это снизило бы порог входа для разработчиков, желающих использовать мемоизацию в своих проектах.

В-третьих, предложенная работа могла бы получить дополнительную ценность, если автор представит сравнительный анализ эффективности методов мемоизации, реализованных в Planning C, с аналогичными подходами в других языках программирования, таких как Python, C++ или Java. Такой анализ предоставит читателям больше данных для оценки конкурентоспособности подхода.

Кроме того, автору стоит уделить внимание созданию библиотеки примеров применения мемоизации в разных областях, таких как вычислительная физика, биоинформатика и экономическое моделирование. Это расширит интерес читательской аудитории и поможет популяризировать предложенный подход.

Наконец, важным направлением является исследование методов адаптивного управления погрешностью мемоизации в условиях нестабильных данных. Например, реализация динамических алгоритмов контроля точности предикторов могла бы обеспечить более надежное применение предложенных методов в задачах с высоким уровнем неопределенности.

В целом, данные рекомендации направлены на дальнейшее развитие исследуемой тематики и ее практическое применение в реальных условиях. Развитие работы в предложенных направлениях укрепит позиции автора в научном сообществе и сделает представленный подход доступным и полезным для более широкой аудитории.

Рекомендую принять статью к публикации без доработок. Она соответствует всем требованиям научного журнала, представляет собой значительный вклад в область программирования и обладает высоким потенциалом для дальнейших исследований.