УДК 004.231.3

П.А. Чернышев., А.Д Фирсов

Днепропетровский национальный университет им. О.Гончара

НАБОР AVR КОМАНД ДЛЯ АППАРАТНОЙ РЕАЛИЗАЦИИ КЛАССИЧЕСКОГО ГА

Розглянуті варіанти автономної цифрової реалізації ГА. З урахуванням мінімізації вартості розв'язку пропонується використання або Risc- мікроконтролерів, або сигнальних цифрових мікроконтролерів. У якості першого кроку при створенні обладнання - розробки логіки, виділяється мінімальний набір команд Atmel-контролера, шляхом оптимізації обчислювальних процедур.

Актуальность проблемы. Современные научные решения характеризуются значительной долей их внедрения. Очень часто фундаментальные алгоритмы становятся регулярно применимыми не подготовленными пользователями. Это достигается за счет инкапсуляции сложной логики и предоставлении пользователю удобного интерфейса. Такой подход применим как для программных, так и для аппаратных решений. Естественно, перед разработчиками возникают задачи не только создания решений, но и их адаптации.

Активное проникновение наукоемких технологий в повседневную жизнь ставит новые задачи перед исследователями, работающими в области оптимизации. При решении соответствующих задач возникает две цели: высокая точность ответа и малая стоимость его получения. Разумеется, достигнуть одновременно наивысших показателей для обоих качеств часто невозможно. Исходя из этого, методы оптимизации специализируются по целям. Для формальных алгоритмов вышеуказанное соотношение склоняется в сторону точности за счет ограничения применимости к различным типам задач. Для эвристических же – на первый план выходит стоимость решения, что важно в технических системах (например, если требуется оптимизировать некоторую функцию на меняющихся входных данных). Рассмотрим далее эвристические алгоритмы, а точнее один из их подклассов – генетические алгоритмы.

В работе исследован не сам метод, а его применение, причем с минимальным участием человека-оператора. Реализация алгоритма в виде отдельного аппаратного модуля позволит не только исключить

ошибку, вносимую человеческим фактором, но и позволит встраивать этот модуль в любую аппаратную систему, где принимаются некоторые решения или выбираются параметры, исходя из некоторых соображений оптимальности.

Актуальность подобных решений огромна, т. к. для автоматизации любого меняющегося во времени процесса необходимо принимать некоторые решения для достижения лучших результатов, причем делать это не только максимально быстро, но и дешево (иначе автоматизация невыгодна). Например, в стиральной машине подобным критерием для лучшего решения может быть минимальный расход воды-электроэнергии/минимальное время стирки; в кофеварке – ароматность напитка/количество кофеина в напитке; в движущейся системе – расчет траектории исходя из наименьшего времени достижения некоторой точки/наименьших трат горючего и т. п.

Примеров существует огромное множество, т. к. с усложнением техники, используемой человеком, постоянно возникает необходимость управлять сложными процессами. Для этого необходима дешевая, как при покупке, так и при эксплуатации, максимально независимая от человека, быстрая аппаратная система, принимающая лучшие решения.

Требования к реализации. Специфика генетических алгоритмов и опыт их применения для решения различных классов задач позволяет говорить, что быстрая, точная, переносимая и универсальная аппаратная реализация ГА позволит решать многие задачи в реальном времени. В данном случае:

– Быстрая реализация – решающая поставленную задачу с минимальными затратами времени и вычислительных ресурсов;

– Переносимая – обладающая возможностью быть использованной в как можно большем количестве задач без значительного изменения ее структуры как аппаратно, так и программно. Наличие данного свойства позволит легко использовать одну и ту же реализацию с малыми изменениями для применения ее в задачах разного рода;

– Универсальная – проявляющая одинаковую способность к решению задач разного рода, оптимизации функций разных классов независимо от исходных данных.

Кроме того, у реализации алгоритма есть такое свойство, как стоимость, под которой подразумевается сумма потребляемых ресурсов во время работы и затрат на изготовление. Из стоимости

также следует такое качество, как минимальная избыточность решения, то есть наличие всех функций, необходимых для решения задачи и отсутствие прочих.

Будем рассматривать автономную цифровую реализацию генетических алгоритмов исходя из следующих предпосылок:

– Цифровая индустрия быстро развивается, стоимость реализации любого алгоритма с использованием вычислительных машин (от микропроцессоров до ЭВМ) будет сравнительно невелика и потенциально будет с каждым годом снижаться;

– Вычислительные мощности цифровых схем с каждым годом растут, при этом вычислительная точность увеличивается;

– Разнообразные аппаратные решения предлагаются большим количеством производителей, каждый из которых предоставляет свой собственный ассортимент, а конкуренция между ними определяет низкую стоимость решения;

– Автономность реализации позволит решать многие задачи, определяя лишь некоторые параметры алгоритма. Автономность также означает возможность получения данных, их обработки и выдачи результатов независимо от человека-оператора.

Очевидно, что универсальная ЭВМ не подходит исходя из вышеуказанных критериев, необходимых для реализации ГА. В самом деле, стоимость ПЭВМ значительна, а уж о минимальной избыточности и говорить не приходится.

Обзор существующих технологий. От недостатка вышеуказанной реализации можно избавиться за счет перехода к программированию микроконтроллерной техники. Микроконтроллеры унифицированы, имеют удобные среды для разработки, обладают огромным потенциалом внедрения в абсолютно разноплановые задачи, они дешевы, имеют энергонезависимую память и, как следствие, малое потребление электроэнергии, что придает им особую мобильность. Но универсальные микроконтроллеры зачастую избыточны. Большинство CISC-микроконтроллеров обладают сотнями команд. Избыточность означает лишнее потребление энергии (и большее тепловыделение), большую стоимость оборудования.

RISC (*Reduced Instruction Set Computing*) — вычисления с сокращённым набором команд. Это философия проектирования процессоров, которая во главу ставит следующий принцип: более компактные и простые инструкции выполняются быстрее. Простая архитектура позволяет, как значительно удешевить процессор, так и

поднять тактовую частоту. Ознакомиться с преимуществами и недостатками, равно как и историей развития можно на сайте [http://cse.stanford.edu/class/sophomore-college/projects-00/risc/].

Возвращаясь к проблеме построения генетического алгоритма на базе микроконтроллера, можем сказать, что, если удастся определить достаточно малый набор команд, необходимый для выполнения ГА, то мы сократим стоимость реализации, ее энергопотребление, не теряя в производительности. Можно выделить команды/последовательности команд, которыми ГА оперирует чаще всего, а также избавиться от разновидностей команд, которые введены для удобства программирования (например, команды вызова подпроцедур и команды возврата). Так мы подходим к следующей идее – использование цифровых сигнальных процессоров.

Цифровой сигнальный процессор (*Digital signal processor*; сигнальный микропроцессор, СМП) – специализированный микропроцессор, предназначенный для цифровой обработки сигналов (обычно в реальном масштабе времени). Основная идея сигнальных микроконтроллеров заключается в реализации команд или их последовательности, необходимых для выполнения конкретного алгоритма аппаратно и удалении команд, незадействованных в вычислительном процессе. При этом – не только значительно увеличивается скорость выполнения данного алгоритма, но и уменьшается объем, занимаемый микропроцессором, уменьшается энергопотребление [3].

Следует заметить, что стоимость цифрового сигнального процессора немного выше, чем соответствующих RISC микропроцессоров. Это объясняется спецификой СМП, т. к. для них характерно значительно большее быстродействие.

Постановка задачи и формулирование требований. Возвращаясь к реализации ГА в СМП, следует отметить необходимость выделения минимума команд, используемых при реализации алгоритма для определения набора команд СМП. Построение сигнального процессора, исполняющего ГА, позволит сохранить малую стоимость или еще ее уменьшить при сохранении, а возможно и росте производительности.

Рассмотрим качества, необходимые для реализации ГА, но уже применительно к СМП или RISC микроконтроллерам:

– Быстрота: микроконтроллер – отдельный физический компонент, занимающийся вычислениями на аппаратном уровне;

– Переносимость: микроконтроллер достаточно просто

применяется к решению любой задачи, т. к. это полностью автономная микросхема. Ее необходимо лишь обеспечить питанием (и, возможно, добавить компонент, вычисляющий фитнесс-функцию);

– Легкость задания фитнесс-функции внешним компонентом или непосредственно программированием микроконтроллера, возможность компоновать выходы одного микроконтроллера с входами другого и, таким образом, генерировать сложные схемы, вычисляющие с большой точностью или производящие многомерную оптимизацию;

– Стоимость подобной схемы мала по сравнению с другими реализациями, стоимость эксплуатации также очень мала (например, энергопотребление на несколько порядков меньше энергопотребления ПЭВМ).

Исходя из вышеприведенных размышлений, можно сформулировать задачу так: необходимо на примере некоторого RISC микроконтроллера (или СМП) выделить необходимый минимальный набор команд, которого будет достаточно для выполнения ГА.

Реализация ГА с точки зрения программирования должна быть:

– Простой, что обеспечивает высокий уровень понимания исходного кода разными программистами и позволяет существенно ускорить процесс разработки за счет распараллеливания процесса написания кода;

– Эффективной, что означает быстроту выполнения;

– Переносимой для удобства отладки и подключения дополнительных источников данных, переносимости на другие архитектуры и пр.

– Распространенной и стандартной, что означает действительную возможность переноса на другие архитектуры с минимальным набором изменений кода программы и возможность легкой отладки на ПЭВМ.

Результаты работы. Опираясь на вышеизложенные требования, для написания ГА был выбран язык C++ (ISO /I EC 14882:1998 – Язык Программирования C++) и один из наиболее корректно поддерживающих данный стандарт компилятор GCC (точнее, его модификация AVR GCC). А для аппаратной реализации микроконтроллер семейства AVR – восьмибитных микроконтроллеров фирмы Atmel [http://www.atmel.com/products/avr/].

В данной реализации ГА популяция представлена массивом хромосом, каждая из которых содержит четное количество байт (здесь

и далее ограничения алгоритма влияют лишь на общее время работы алгоритма, при этом на необходимый набор команд они не влияют). В хромосоме также хранится значение фитнесс-функции. Алгоритм ГА состоит из создания начальной популяции (популяция генерируется случайно), потом пересчитывается значение фитнесс-функции. Затем в цикле (заданное количество итераций) производится мутация, кроссовер, селекция. Заданием количества итераций можно определять время остановки алгоритма.

Операция селекции также имеет ряд особенностей, упрощающих ее выполнение и обеспечивающих свойство элитарности, например, лучшая хромосома сохраняется, если полученная популяция не содержит хромосому лучше.

```
ga_chromosome *genetic_algorithm_classic(const uint8_t iterations)
{
  ga_population pop; pop.generate();
  for (uint8_t i=0; i < iterations; i++) {
    pop.mutate();
    pop.crossover();
    pop.select();
  }
  return pop.end_of_population;
}
```

Алгоритм мутации оперирует не байтами, а битами. Это позволяет увеличить его эффективность, если в контроллере не введены дополнительные операции по работе с битами в середине байтового массива. Ресурсоемкими являются генерация псевдослучайной величины и (предполагаемая) работа с битами в середине байтового массива.

```
void mutate() {
   uint16_t r;
   memcpy(end_of_population, end_of_population-1, sizeof(ga_chromosome));
   //chromosome – массив все хромосом
   for (chrom_iter = chromosome; chrom_iter < end_of_population; chrom_iter++) {
     char_iter1 = chrom_iter->data;
     char_iter2 = chrom_iter->data + MAX_CAPACITY; //MAX_CAPACITY – размер хромосомы в байтах
     while (char_iter1 != char_iter2) {
```

```
        r = rand_generator::rand16(); //получение псевдослучайных 16 бит
        if ((r & 0xff) < PMUTATION_BIN) *char_iter1 = r >> 8;
        char_iter1++;
      }}
  }
```

В течение кроссовера скрещиваются наилучшие особи. В совокупности с запоминанием лучшего элемента в алгоритме мутации это позволяет увеличить скорость схождения алгоритма, хотя и способствует более быстрому вырождению популяции при большом количестве итераций. Ресурсоемкими являются операции сортировки и генерации псевдослучайной величины.

```
  void crossover()
  {
   for (chrom_iter = chromosome; chrom_iter <= end_of_population; chrom_iter++)
    calculate_fitness(chrom_iter);
   //сортировка по возрастанию значения фитнесс-функции
   qsort(chromosome, popSize, sizeof(ga_chromosome), fitness_comparing);
   chrom_iter = end_of_population - 1;
   //восстанавливаем лучшую особь
   if (end_of_population->fitness > chrom_iter->fitness)
    memcpy(chrom_iter,end_of_population, sizeof(ga_chromosome));
   //SELECTION_NUMBER = (P_S * Размер_популяции / 100)/2 – количество новых хромосом
   ga_chromosome *iter2 = chrom_iter - (SELECTION_NUMBER << 1);
   while (chrom_iter != iter2)
   {
    //одноточечный кроссовер 2 хромосом
    crossover((uint8_t *)chrom_iter, (uint8_t *)(chrom_iter - 1));
    chrom_iter -= 2;
   }
   popSize += SELECTION_NUMBER;
  }
```

Алгоритм селекции сильно упрощен, однако это не влияет на наборы команд и их последовательностей. Ресурсоемкими продолжают быть операции сортировки и генерации псевдослучайной величины. Также необходимо уделить внимание операциям

умножения/деления. При наличии их аппаратной реализации это позволит ускорить не только текущий вариант селекции, но также и классический вариант – «рулетку».

```
void select()
{
 //сортировка по возрастанию значения фитнесс-функции
 qsort(chromosome, popSize, sizeof(ga_chromosome), &fitness_comparing);
 //временные переменные для случайных величин
 uint16_t r;
 uint8_t r1, r2;
 //тип population_size предназначен для итерации массива, он учитывает размер популяции
 population_size i = 0;
 popSize--; //текущий размер популяции
 uint16_t psize = POP_SIZE - 1; //ожидаемый размер популяции
 while (i != psize)
 {
  r = rand_generator::rand16();
  r1 = (((r & 0xff) * (popSize - i)) >> 8) + i;
  r2 = (((r >> 8) * (popSize - r1)) >> 8) + r1;
  chromosome[i++] = chromosome[r2];
 }
 chromosome[i] = chromosome[popSize];
 end_of_population = chromosome + i;
 popSize = POP_SIZE;
}
```

Для генерации случайных чисел предлагается ввести аппаратную реализацию, что поможет сильно улучшить эффективность операций генерации, мутации и отбора (предлагается использовать алгоритм LFSR, имеющий простую и быструю аппаратную реализацию [6], [www.newwaveinstruments.com/ resources/articles/m_sequence_linear_feedback_shift_register_lfsr.htm]). Повышение эффективности можно легко заметить при рассмотрении полученного ассемблерного кода. Получение случайного числа методом LFSR:

```
x <<= 1;
91800104   LDS    R24,0x0104      Loading X
91900105   LDS    R25,0x0105
```

```
0F88      LSL    R24            Logical Shift Left
1F99      ROL    R25            Rotate Left Through Carry
93900105  STS    0x0105,R25     Storing X
93800104  STS    0x0104,R24
if (x<0) {
2399      TST    R25            Test for Zero or Minus
F46C      BRGE   PC+0x0E        Branch if greater or equal, signed
x ^= 0xD002; //для 16-и битного числа
019C      MOVW   R18,R24        Copy register pair
E082      LDI    R24,0x02       Load immediate
ED90      LDI    R25,0xD0       Load immediate
2782      EOR    R24,R18        Exclusive OR
2793      EOR    R25,R19        Exclusive OR
93900105  STS    0x0105,R25     Storing new X
93800104  STS    0x0104,R24
}
```

Генерация псевдослучайного числа таким образом выполняется за 10 – 30 тактов. Встроенная операция генерации выполняется за 1450 тактов.

Аналогичная ситуация имеет место и для аппаратной поддержки сортировки и/или слияния сортированных последовательностей (увеличение эффективности операций скрещивания и отбора). Для улучшения качества работы операций мутации рекомендуется ввести операции для быстрой работы с битами внутри байтового массива.

На эффективность алгоритма сильно влияют коэффициенты мутации и скрещивания, размер популяции. Размер популяции выбираем таким образом, чтобы с учетом увеличения массива хромосом во время кроссовера, индекс массива не превысил 256 элементов, что ускоряет индексацию и проход по массиву. Размер особи задается константой до компиляции, выражается в количестве байт в хромосоме. Фитнесс-функция также задается заранее. Впрочем, при ее реализации, можно использовать входы/выходы микроконтроллера, что позволит ее реализовать внешними компонентами.

Исходный код программы обработан компилятором AVR-GCC 4.3 в режимах оптимизации по скорости выполнения и размеру кода. Исходя из этих результатов, была составлена таблица встречаемости команд. Таблица сформирована из ассемблерного листинга программы.

Таблица 1

Частоты встречаемости команд при компиляции в режиме O3 (gcc 4.3)

Команда	Раз встречается	Тиков выполняется	Предполагается заменить на:
Movw	231	1	
Ldi	112	2	Ldd
Subi	100	2	Sbc
Adc	99	1	
Ldd	93	2	
Add	92	1	Adc
Rjmp	88	2	
Sbci	86	1	Sbc
Std	71	2	
Ld	58	2	Ldd
Eor	56	1	
St	53	2	Std
Mov	50	1	Movw
Cpc	48	1	Cp
Adiw	46	2	Adc
Cp	44	1	
Jmp	39	4	Rjmp
Push	36	2	
Brne	35	2	
Mul	34	2	
Cpi	28	1	Cp
Sbiw	27	2	Sbc
Sbc	26	1	
Sub	22	1	Sbc
Brcc	19	2	Sbrs
Call	19	4	Rjmp
Pop	18	2	
Sts	18	3	Std
Put	17	1	
Ret	14	4	Rjmp
Sbrc	14	2	Srbs
Com	13	1	
Breq	12	2	Brne
In	12	1	

Окончание табл. 1

Команда	Раз встречается	Тиков выполняется	Предполагается заменить на:
Lds	12	2	Ldd
Brcs	11	2	Sbrs
Brlt	11	2	
Andi	10	1	And
Icall	9	3	Rjmp
Sec	9	1	Bst
Sbrs	7	2	
Cli	5	1	(можно обойтись без прерываний)
Brge	4	2	Brlt
Rcall	4	3	Rjmp
Dec	3	1	Sbc
Neg	3	1	
Or	3	1	
And	2	1	
Ijmp	2	2	Rjmp
Lpm	2	3	(можно не пользоваться загрузкой кода)
Lsr	2	1	
Ror	2	1	Or
Brtc	1	2	Sbrs
Bst	1	1	
Elpm	1	4	(можно не пользоваться загрузкой кода)
Inc	1	1	Adc

Выводы. Отбрасывание схожих по функциональности операций дает возможность сделать вывод, что **ГА возможно реализовать всего лишь при использовании набора из 22 операций.** Причем ясно, что некоторые из оставленных операций могут быть выражены через другие, например, sbc (вычитание) или mul (умножение), но дальнейшее отбрасывание операторов будет приводить к колоссальному росту требований на вычислительную мощность.

Было отмечено, что дешевая автономная цифровая реализация генетического алгоритма нивелирует его недостатки вследствие возможности активного применения ее к различным задачам (из-за малой стоимости), достаточно быстрого нахождения оптимума (за счет использования современных наработок в области цифровой техники) и отсутствия постоянного вмешательства со стороны человека-оператора (а как следствие и возможность выполнения пакетных расчетов).

Рассмотренные варианты автономной цифровой реализации ГА с учетом минимизации стоимости сводятся к использованию либо RISC-микроконтроллеров, либо сигнальных цифровых микроконтроллеров, причем вторые предпочтительнее вследствие более высокого быстродействия, меньшего размера и т. п.

Для использования СМП необходимо выделить минимальный набор команд, построенных аппаратно, для реализации генетического алгоритма.

Программа была скомпилирована, а ее ассемблерный листинг и «hex-прошивка» построены для микроконтроллера AVR ATMega128. Выбор сделан исходя из наличия в данном контроллере команд для работы над 16-битными регистрами, аппаратного умножения и быстрого относительного перехода. Впрочем, это не означает, что невозможно или тяжело получить результаты данной работы для другого типа микропроцессора. Код реализации генетического алгоритма написан согласно стандарту ISO/IEC 14882:1998 и может быть повторно использован во всех совместимых с ним компиляторах. Необходимо лишь учесть другие аппаратно-зависимые части.

Полученный набор команд состоит из 22 допускает еще сокращение за счет значительных трат вычислительных ресурсов (например, возможно исключение команд mul, sbc, brne, eor).

Рекомендуется введение следующих аппаратных возможностей микроконтроллера:

- сортировка элементов в памяти;
- обмен значений двух ячеек в памяти;
- слияние двух отсортированных последовательностей;
- генерация случайного числа;
- обмен двух элементов в памяти;
- инструмент для быстрого прохождения массива данных в памяти;

Отметим несколько идей при использовании микроконтроллеров, вычисляющих ГА:

1. Можно распараллелить процесс оптимизации (или увеличить точность) при перенаправлении вывода данных с нескольких микроконтроллеров на ввод одного, оптимизирующего полученную последовательность.
2. Также возможно проводить параллельную многомерную оптимизацию, соединяя выводы микроконтроллеров, оптимизирующих функцию по одной из координат с микроконтроллером, проводящим оптимизацию функции по другой координате.
3. Очевидно, что можно реализовать фитнесс-функцию внешним образом, **значительно** увеличив скорость оптимизации за счет использования внешнего вычисляющего устройства. Для СМП этот пункт может быть не так актуален, вследствие возможности встраивания в него модуля вычислений с плавающей запятой.

Библиографические ссылки

1. **Holland J.H.** Adaptation in Natural and Artificial Systems. Ann Arbor: The University of Michigan Press, 1975. –218 p.
2. **Randy L. Haupt, Sue Ellen Haupt.** Practical genetic algorithms. –2nd ed. John Wiley & Sons, 2004. –236 p.
3. **Солонина А.** Алгоритмы и процессоры цифровой обработки сигналов. / А. Солонина, Д. Улахович, Л. Яковлев. – С-Пб., 2002 –464 с.
4. **Кнут Д.** Искусство программирования. 2. Получисленные алгоритмы. – М., 2005 –832 с.

Надійшла до редколегії 27.08.08