ТОЧКА ЗРЕНИЯ

УДК 681.3

О НЕКОТОРЫХ СТЕРЕОТИПАХ ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ

Г. В. Байдин
(РФЯЦ-ВНИИТФ)

Проверяется состоятельность нескольких рабочих идей параллельного программирования. Рассмотрены вопросы счета на фоне обменов, оптимального размера сообщений, оптимальности неблокирующих и коллективных операций.

Введение

Занимаясь программированием реальных параллельных численных методик уже свыше десяти лет, автор данной публикации постепенно стал осознавать, что многие представления о параллельности, распространенные в среде разработчиков-программистов, живут как идеи, не имея под собой оснований. Некогда возникнув в виде задумок разработчиков, они либо так и не были внедрены, либо уже отжили свой век, но тем не менее и поныне смущают умы. Развенчиванию некоторых из таких стереотипов параллельного программирования посвящена данная работа. Основная доля сказанного будет касаться системы MPI.

Итак, в среде программистов бытуют следующие мнения:

1. *Обмены выгодно проводить на фоне счета.* Идеями совмещать вычисления и обмены пронизан весь стандарт MPI. Идеи эти базируются на том, что процессор плюс сетевой адаптер — сложная составная система, отдельные части которой могут функционировать независимо, и грех этим не воспользоваться.
2. *Существует оптимальный размер сообщений.* Если смотреть на чужую программу со стороны как на потоки данных, перетекающих от процесса к процессу, невольно замечаешь, насколько неоптимально организованы эти потоки. Казалось бы, достаточно перестроить поток сообщений так, чтобы длины сообщений были оптимальными, и можно существенно ускорить обмены в программе.
3. *При необходимости одновременно проводить множественные обмены операционная система (ОС) сама оптимально с ними справится, если обмены сделать неблокирующими.* Такие обмены полезны прежде всего, когда трудно вручную корректно составить очередь из множества процессов, "желающих" обмениваться. Но при этом очень хотелось бы положиться на "интеллект" ОС, считая, что она выполнит все эти множественные обмены оптимально.
4. *Коллективные операции не только удобны, но и оптимальны.* Так утверждают прежде всего производители программного обеспечения (ПО) для параллельного счета, рекламируя свои разработки.
5. *Добавление каждого процессора всегда, хоть на немного, ускоряет расчет.* Идея эта укоренилась буквально на уровне подсознания и особенно популярна у руководителей, мыслящих стратегически при дележе вычислительных ресурсов.

Каждому из этих стереотипов — *мифов параллельности* — посвящен отдельный раздел статьи.

Надо признаться, что, приступая к этой работе, автор не имел четкого плана, о чем и как писать. Было только желание выразить удивление, благо поводов для этого имеющаяся под рукой реализация MPI предоставила немало. Примеры, на которых проверялись все сомнения, конструировались тут же, далеко не в оптимальном виде. Поэтому во многих случаях полученные выводы вполне могут основываться на заблуждениях самого автора: не так была понята документация или в программу вкралась ошибка. Именно поэтому автор посчитал нужным привести тексты примеров — все заблуждения, и авторские тоже, должны быть выявлены.

Особо следует подчеркнуть, что результаты всех проведенных экспериментов, как положительных, так и отрицательных, в значительной мере относительны: другие настройки, другая элементная часть, другие версии ПО дадут иные результаты. Автор же призывает удивиться тому, как неожиданно может себя вести вроде бы знакомый пользователю инструмент и насколько зыбкими могут оказаться сложившиеся стереотипы.

1. О выгодности проведения обменов на фоне счета

Мысль о том, что обмены выгодно проводить на фоне счета, настойчиво внедряется в головы пользователей во многих руководствах по параллельности. Вот, к примеру, что пишется в книге Воеводиных [1, с. 285] по поводу полезности приема/передачи без блокировки: "...На фоне дальнейшего выполнения программы одновременно происходит и обработка асинхронно запущенной операции. В принципе данная возможность исключительно полезна для создания эффективных программ. В самом деле, программист знает, что в некоторый момент ему потребуется массив данных, который вычисляет другой процесс. Он заранее выставляет в программе асинхронный запрос на получение данного массива, а до того момента, когда массив реально потребуется, он может выполнять любую другую полезную работу...". После такого воодушевляющего вступления совершенно непонятыми остаются далее идущие строки: "...далеко не всегда асинхронные операции эффективно поддерживаются аппаратурой и системным окружением. Поэтому не стоит удивляться, если эффект ... окажется нулевым".

Давайте посмотрим сами, каков эффект асинхронности на имеющемся в наличии вычислительном комплексе для маленькой программы из двух процессов, обменивающихся на фоне счета.

Контрольный вариант, в котором сначала происходит расчет массива данных, а затем весь массив передается на другой процесс одной операцией, имеет следующий вид:

```
ndrop=1; sizi=siza/ndrop;
t0 = MPI_Wtime();
for(j=0;j<max_it;j++){ // global iteration:
  nBeg=0;cout<<"A2- "<< MPI_Wtime()-t0 << endl;
  for(k1=0;k1<ndrop;k1++){ // drop iteration:
    for(k=0;k<sizi;k++){ // drop iteration:
      ij=nBeg+k;
      My_Value[ij] = cos(ij*1.0e-10)*Gh_Value[ij]*cos(ij*1.0e-9)+
                     sin(ij*1.0e-10)*Gh_Value[ij]*sin(ij*1.0e-9);
    };
    MPI_Isend(My_Value+nBeg, sizi, MPI_DOUBLE, Targ, 1, MPI_COMM_WORLD, reqst+(2*k1));
    MPI_Irecv(Gh_Value+nBeg, sizi, MPI_DOUBLE, Targ, 1, MPI_COMM_WORLD, reqst+(2*k1+1));
    nBeg += sizi;
 };
 MPI_Waitall(ndrop*2, reqst, astatus);
};
```

Экспериментальный же вариант заключается в том, что массив данных обрабатывается фрагментами (использовалось 10 фрагментов). По окончании расчета каждого фрагмента последний отсылается неблокирующей передачей ожидающему соседу, а самому процессу предлагается, не мешкая,

заняться расчетом следующего фрагмента. Текст кода экспериментального варианта отличается только значением числа фрагментов: `ndrop=10;`.

Итоги эксперимента получены с помощью замера времени работы программ: оно оказалось совершенно одинаковым. Таким образом, можно считать, что данный вычислительный комплекс не позволяет рассинхронизовывать вычисления и обмены в рамках одного процесса. Положение не изменилось и после некоторых попыток привлечь переменные окружения.

Это был первый из развенчанных мифов, с которым автор столкнулся три года назад и с которого он начал создавать свою "коллекцию" мифов. При подготовке материала к докладу в 2007 году появилась идея провести расчет заново на новой технике, чтобы иметь "свежие" цифры. Каково же было удивление, когда тестовый вариант программы просчитался вдвое быстрее контрольного! Изменился комплекс, изменились версии ПО, и в конце концов идея возможности фонового обмена оказалась подтверждена. Одно только не дает радоваться в полной мере: в другом месте, на других комплексах может сложиться ситуация трехлетней давности.

2. О существовании оптимального размера сообщений

Следующий миф гласит: разбивая сообщения на части оптимальной длины, можно существенно сократить время передачи больших массивов. Говоря по-другому, вычислительная система настолько неодинаково отрабатывает передачу сообщений разной длины, что на этом можно получать существенный выигрыш. Например, в одной из диссертаций учет разницы в скорости передачи данных порциями разного размера положен в основу протокола организации согласованной работы процессов на распределенной вычислительной системе.

Проверить важность этого положения для единой многопроцессорной системы достаточно просто. Возьмем большой массив чисел типа `real*8` и станем его передавать с одного процесса на другой порциями (число порций от 1 до 134 217 728, длина порции (в элементах массива) соответственно от 134 217 728 до 1, или от 1 Гбайта до 8 байт). Фрагмент программного кода следующий:

```
    if(my_proc==0){
      for(i_cnt=1;i_cnt<=max_cnt;i_cnt++){
      for(is=0;is<max_is;is++){
        n_p=pow(2,is); size_p= dim/n_p;
        MPI_Barrier(MPI_COMM_WORLD); t0 = MPI_Wtime();
        for(ip=0;ip<n_p;ip++)
MPI_Send(A+(ip*size_p),size_p,MPI_DOUBLE,1,0,MPI_COMM_WORLD);
        tip = MPI_Wtime() - t0;
        Res[is][0]=i_cnt; Res[is][1] += tip; Res[is][2] += (tip*tip);
        Res[is][3]=Res[is][1]/i_cnt; Res[is][4]=Res[is][2]/i_cnt;
        Res[is][5] = 2.0*sqrt(Res[is][4] - Res[is][3]*Res[is] [3]);
cout<< n_p <<" * "<< size_p << {"<< Res[is][3]<<", "<< Res[is][5] <<"} "<< endl;
      }}; };
    if(my_proc==1){
      for(i_cnt=1;i_cnt<=max_cnt;i_cnt++){
      for(is=0;is<max_is;is++){
        n_p=pow(2,is); size_p= dim/n_p;
        MPI_Barrier(MPI_COMM_WORLD); t0 = MPI_Wtime();
        for(ip=0;ip<n_p;ip++)
MPI_Recv(A+(ip*size_p),size_p,MPI_DOUBLE,0,0,MPI_COMM_WORLD,&status);
        tip = MPI_Wtime() - t0;
        Res[is][0]=i_cnt; Res[is][1] += tip; Res[is][2] += (tip*tip);
        Res[is][3]=Res[is][1]/i_cnt; Res[is][4] =Res[is][2]/i_cnt;
        Res[is][5] = 2.0*sqrt(Res[is][4] - Res[is][3]*Res[is][3]);
cout<<n_p<<" * "<<size_p<< {"<<Res[is][3] <<", "<<Res[is][5]<<" } "<<endl;
      }}; };
```

Табл. 1 содержит результаты замеров времени (в секундах) вместе со статистической погрешностью, отдельно для посылающей и принимающей сторон. Если есть предпочтительные размеры передаваемых сообщений, то полученная зависимость должна иметь минимум где-нибудь внутри интервала длин. Рассматривая результаты, содержащиеся в таблице, приходим к выводу, что нет особого смысла искать оптимальные размеры сообщений: современная вычислительная система, по-видимому, сама выбирает наиболее оптимальный с ее точки зрения способ передачи нужного объема данных, и не следует ей в этом мешать.

Таблица 1

Время посылки и приема сообщений

Число порций	Размер порции	Время посылки, с	Время приема, с
1	134 217 728	1,205 ± 0,010	1,205 ± 0,010
2	67 108 864	1,204 ± 0,008	1,205 ± 0,008
4	33 554 432	1,206 ± 0,010	1,207 ± 0,010
8	16 777 216	1,201 ± 0,009	1,202 ± 0,009
16	8 388 608	1,196 ± 0,014	1,197 ± 0,014
32	4 194 304	1,186 ± 0,016	1,187 ± 0,017
64	2 097 152	1,166 ± 0,016	1,167 ± 0,017
128	1 048 576	1,107 ± 0,020	1,107 ± 0,020
256	524 288	1,022 ± 0,016	1,023 ± 0,016
512	262 144	0,987 ± 0,012	0,988 ± 0,012
1 024	131 072	0,984 ± 0,010	0,985 ± 0,010
2 048	65 536	0,993 ± 0,007	0,993 ± 0,007
4 096	32 768	1,026 ± 0,006	1,026 ± 0,006
8 192	16 384	1,083 ± 0,011	1,083 ± 0,011
16 384	8 192	1,187 ± 0,014	1,187 ± 0,014
32 768	4 096	1,384 ± 0,020	1,384 ± 0,020
65 536	2 048	1,735 ± 0,051	1,735 ± 0,051
131 072	1 024	1,540 ± 0,932	1,540 ± 0,932
262 144	512	2,212 ± 0,031	2,242 ± 0,025
524 288	256	2,462 ± 0,040	2,499 ± 0,029
1 048 576	128	3,069 ± 0,118	3,094 ± 0,109
2 097 152	64	4,497 ± 0,243	4,524 ± 0,240
4 194 304	32	6,102 ± 0,192	6,118 ± 0,197
8 388 608	16	10,534 ± 0,392	10,560 ± 0,373
16 777 216	8	17,799 ± 1,469	17,815 ± 1,480
33 554 432	4	18,645 ± 0,254	18,645 ± 0,254
67 108 864	2	42,448 ± 4,013	42,448 ± 4,013
134 217 728	1	84,910 ± 3,474	84,926 ± 3,464

3. Об оптимальном выполнении системой неблокирующих обменов

"На многих системах возможно улучшить производительность работы, если организовать обмены и счет с перекрытием друг друга. ...Альтернативный механизм, который часто способен дать лучшую производительность, — использование неблокирующего обмена" [2]. Попробуем, следуя описанию стандарта MPI, проникнуться элегантностью и мощью MPI и построить пример, демонстрирующий обещанное повышение производительности. Речь здесь пойдет не о перекрытии вычислений и обменов (об этом говорилось в разд. 1), а о способности системы воспользоваться свободой неблокирующих сообщений и оптимально передать порожденное множество таких сообщений, получив выигрыш во времени.

Возьмем близкую многим проблему геометрической декомпозиции: предлагается обеспечить обмен большим объемом данных между соседствующими блоками, на которые разбита прямоугольная счетная область задачи (рис. 1). У каждого блока есть *свои* данные, которые нужно раздать всем соседям, имеющим с текущим блоком хотя бы одну общую граничную точку. В ответ от тех же соседей нужно получить их данные. Общая формула размера своих данных взята такой:

$$\mathtt{my_size} = \mathtt{NN} + \mathtt{my_proc} * 100,$$

где $NN = \{10\,000,\ 5\,000,\ 2\,500,\ 1\,000,\ 500,\ 250,\ 100\}$.

2	6	9
1	5 4	8
0	3	7

Рис. 1. Разбиение прямоугольной счетной области

В результате и количество соседей, и количество получаемых/передаваемых данных у каждого блока разное. Разными предполагаются и времена передач. Поэтому, ожидая существенной рассинхронизации в обменных операциях, очень заманчиво использовать неблокирующие передачи, запустить разом все обмены и ждать, пока система выполнит их в том порядке, в каком ей удобнее.

Перед выполнением обменов разными способами в программе имеется фрагмент, определяющий адреса соседей и длины ожидаемых от них сообщений (время исполнения этого фрагмента в сравнительное таймирование не входило). Фрагмент обменов в тестовом варианте имел следующий вид:

```
// ТЕСТОВЫЙ ВАРИАНТ ОБМЕНОВ
for(i=0;i<count_L;i++) {
  MPI_Isend(My_Value,           my_size,MPI_DOUBLE,nei_L[i],0,COMM,reqst+i);
  MPI_Irecv(Gn_Value+dis_L[i],lng_L[i],MPI_DOUBLE,nei_L[i],1,COMM,reqst+(i+count_L));
}; num_reqs = 2*count_L;
for(i=0;i<count_R;i++) {
  MPI_Isend(My_Value,           my_size,MPI_DOUBLE,nei_R[i],1,COMM,reqst+(num_reqs+i));
  MPI_Irecv(Gh_Value+dis_R[i],lng_R[i],MPI_DOUBLE,nei_R[i],0,COMM,
      reqst+(num_reqs+i+count_R));
}; num_reqs += 2*count_R;
if(count_B) {
  MPI_Isend(My_Value,           my_size,MPI_DOUBLE,nei_B[0],2,COMM,reqst+(num_reqs));
  MPI_Irecv(Gn_Value+dis_B[0],lng_B[0],MPI_DOUBLE,nei_B[0],3,COMM,reqst+(num_reqs+1));
  num_reqs += 2;
};
if(count_T) {
  MPI_Isend(My_Value,           my_size,MPI_DOUBLE,nei_T[0],3,COMM,reqst+(num_reqs));
  MPI_Irecv(Gh_Value+dis_T[0],lng_T[0],MPI_DOUBLE,nei_T[0],2,COMM,reqst+(num_reqs+1));
  num_reqs += 2;
};
MPI_Waitall(num_reqs, reqst, astatus);
```

В качестве контрольного варианта взята программа, использующая те же обменные операции, но по очереди:

```
// КОНТРОЛЬНЫЙ ВАРИАНТ
  for(i=0;i<4;i++) {
    if(i<count_L) Targ=nei_L[i]; else Targ=MPI_PROC_NULL;
    if(i<count_R) Sour=nei_R[i]; else Sour=MPI_PROC_NULL;
    MPI_Isend(My_Value,           my_size, MPI_DOUBLE, Targ, 0, COMM, reqst );
    MPI_Irecv(Gh_Value+dis_R[i], lng_R[i], MPI_DOUBLE, Sour, 0, COMM, reqst+1);
    MPI_Waitall(2, reqst, astatus);
    MPI_Isend(My_Value,          my_size, MPI_DOUBLE, Sour, 1, COMM, reqst );
    MPI_Irecv(Gn_Value+dis_L[i], lng_L[i], MPI_DOUBLE, Targ, 1, COMM, reqst+1);
    MPI_Waitall(2, reqst, astatus);
  };
    if(0<count_B) Targ=nei_B[0]; else Targ=MPI_PROC_NULL;
    if(0<count_T) Sour=nei_T[0]; else Sour=MPI_PROC_NULL;
    MPI_Isend(My_Value,            my_size, MPI_DOUBLE, Targ, 2, COMM, reqst );
    MPI_Irecv(Gh_Value+dis_T[0], lng_T[0], MPI_DOUBLE, Sour, 2, COMM, reqst+1);
    MPI_Waitall(2, reqst, astatus);
    MPI_Isend(My_Value,            my_size, MPI_DOUBLE, Sour, 3, COMM, reqst );
    MPI_Irecv(Gh_Value+dis_B[0], lng_B[0], MPI_DOUBLE, Targ, 3, COMM, reqst+1);
    MPI_Waitall(2, reqst, astatus);
```

Результаты сравнения этих двух способов обмена (время в секундах) на разном количестве данных, от 100 до 10 000 чисел в каждом блоке, приведены в табл. 2. Основной вывод — на больших объемах *упорядоченные* операции Isend+Irecv выполняются в 5—10 раз быстрее, чем они же, но "сваленные в кучу".

Таблица 2

Время выполнения передачи данных "с общим стартом" и по очереди (с)

Кол-во чисел	С "общим стартом"	По очереди
10 000	0,191	0,023
5 000	0,067	0,012
2 500	0,049	0,007
1 000	0,025	0,004
500	0,699	0,003
250	0,013	0,002
100	0,006	0,002

4. Об оптимальности операций коллективного обмена

"Операции коллективного обмена, помимо удобств пользования, всегда оптимальны",— эта логичная мысль всячески поддерживается как зарубежными, так и отечественными производителями параллельного ПО. Неоднократно с их стороны автору доводилось слышать о ведущихся серьезных работах по оптимизации стандартных процедур обмена. Выясним, каков же в действительности может быть выигрыш использования коллективных операций по сравнению с индивидуальными. Например, определим, за какое время один процесс сможет раздать 1 Гбайт данных N процессам при помощи коллективной операции MPI_Bcast и сколько времени это будет выполняться при простом обмене.

Вариант теста:

```
MPI_Barrier(MPI_COMM_WORLD); t0 = MPI_Wtime();
MPI_Bcast(A, dim, MPI_DOUBLE, 0, MPI_COMM_WORLD);
tip = MPI_Wtime() - t0;
```

Контрольный вариант программы на основе простой конвейерной передачи массива порциями, количество которых равно числу процессов без единицы:

```
n_p =(total_procs_size - 1); size_p= dim/n_p;
my_pred=my_proc-1;my_next=my_proc+1;
if(my_pred<0)                 my_pred=MPI_PROC_NULL;
if(my_next>=total_procs_size)my_next=MPI_PROC_NULL;
np_times=2*n_p - 1;
MPI_Barrier(MPI_COMM_WORLD); t0 = MPI_Wtime();
for(ip=0;ip<np_times;ip++) {
ips=ip-my_proc; ipr=ips+1;
if((ips>=0)&&(ips<n_p)){s_d=ips*size_p; ip_next=my_next;}
else {s_d=0;ip_next=MPI_PROC_NULL;}
if((ipr>=0)&&(ipr<n_p)){r_d=ipr*size_p; ip_pred=my_pred;}
else {r_d=0;ip_pred=MPI_PROC_NULL;}
  MPI_Sendrecv(A+s_d,size_p,MPI_DOUBLE,ip_next,0,
               A+r_d,size_p,MPI_DOUBLE,ip_pred,0,
                                 MPI_COMM_WORLD,&status);
};               tip = MPI_Wtime() - t0;
```

Сравнение результатов, приведенных в табл. 3, 4, показывает, что с помощью простых обменов, организованных, к примеру, совмещенным Send+Recv, та же работа выполняется ничуть не хуже.

Таблица 3

Время выполнения обмена с помощью операции MPI_Bcast

N	2	4	8	16	32	64	128
t, с	1,76	3,50	5,94	6,29	6,88	14,13	15,56

Таблица 4

Время выполнения обмена с помощью операции MPI_Sendrecv

N	2	4	8	16	32	64	128
t, с	1,05	2,94	4,52	5,37	5,64	5,85	6,14

5. Об ускорении расчета при добавлении процессора

"В программах, написанных для "плавающего" количества процессоров, каждый добавляемый процессор хотя бы на чуть-чуть ускоряет вычисления. Пределом масштабируемости такой программы становится ситуация, когда очередной процессор настолько же сокращает объем работ, насколько привносит накладных расходов." Весьма популярно заблуждение, что эта мысль есть правило, а не редкое исключение. Увы, это не так, по крайней мере для разностных методик того спектра задач математической физики, который исследуется в атомной отрасли. Речь здесь идет уже не о MPI,

а о способах декомпозиции конечномерных задач. Автор первый раз обнаружил свое заблуждение по этому поводу, когда проводил исследования масштабируемости своей параллельной программы, полагая, что изучает эффективность работы многопроцессорных систем.

Предположим, что рассчитывалась некоторая двумерная задача, например теплопроводности, на регулярной сетке из $1\,025 \times 1\,025$ ячеек по некоторой гипотетической программе на разном количестве процессоров. Пусть соответствующие временные замеры на графике зависимости ускорения расчета от числа процессоров выглядят так, как показано на рис. 2 (пунктир используется для их тривиального соединения).

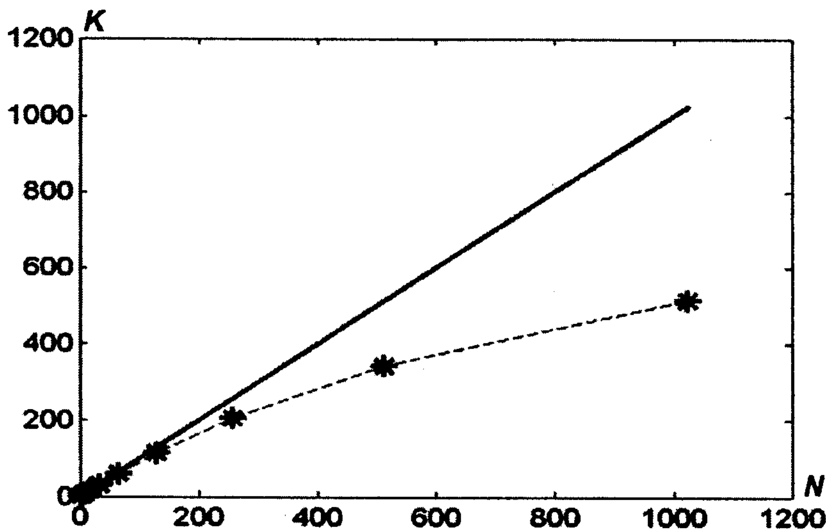

Рис. 2. Зависимость ускорения расчета от числа процессоров: * — результаты замеров; — — идеальное ускорение

За интерпретацией результата обратимся к оценочной формуле зависимости коэффициента ускорения расчета K от числа используемых процессов N:

$$K(N) = \frac{t_{\text{посл}} + t_{\text{парал}}}{t_{\text{посл}} + t_{\text{парал}}/N + t_{\text{обмен}}}. \tag{1}$$

Можно предположить, что задача имеет либо большую долю последовательных вычислений, либо значительное время обменов. Ну а если эти результаты получены для *идеальной* программы, в которой вовсе нет последовательного участка кода ($t_{\text{посл}} = 0$), а размер всех обменов пренебрежимо мал ($t_{\text{обмен}} = 0$), то можно смело предполагать, что сам вычислительный комплекс начинает терять эффективность на числе процессоров, начиная со 150, поскольку тогда оценка (1) будет иметь график идеального ускорения (см. рис. 2).

Однако, скорее всего, дело в другом. Если для распараллеливания применялась геометрическая декомпозиция по столбцам, то с учетом вышепринятых упрощений необходимо пользоваться не оценкой (1), а ее разновидностью, содержащей функцию взятия целой части:

$$K(N) = \frac{n}{\left[\frac{n}{N}\right] + 1}. \tag{2}$$

Но график оценки (2) выглядит принципиально иным образом (рис. 3). Из его анализа можно сделать два важных вывода: 1) отклонения от линии идеального ускорения есть и в идеальных с точки зрения программы условиях; 2) на количестве процессоров существуют интервалы, в которых вообще нет роста ускорения расчета. Тем самым вычислительный комплекс "оправдан": никакого ухудшения эффективности в аппаратной части не было.

Таким образом, последний из рассматриваемых мифов также опровергнут.

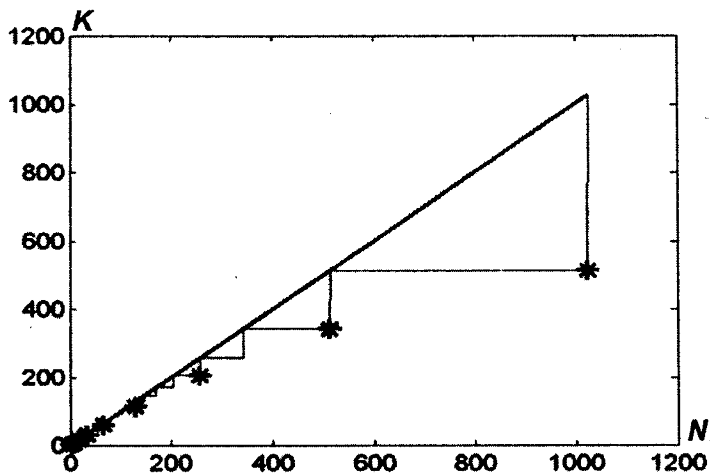

Рис. 3. Зависимость ускорения расчета от числа процессоров: * — результаты замеров; — — идеальное ускорение; ······ — оценка (2)

Заключение

Итак, на пяти примерах была проиллюстрирована основная мысль: в параллельном программировании сложилось множество стереотипов. Автор призывает периодически проводить их ревизию: возможно, какой-то из них уже перестал отражать действительность и мешает правильно воспринимать реальность.

Список литературы

1. *Воеводин В. В., Воеводин Вл. В.* Параллельные вычисления. СПб.: БХВ-Петербург, 2002.

2. MPI: A Message-Passing Interface Standard. June 12, 1995. http//:www-unix.mcs.anl.gov/mpi.

Статья поступила в редакцию 09.07.07.