А.В.Коновалов

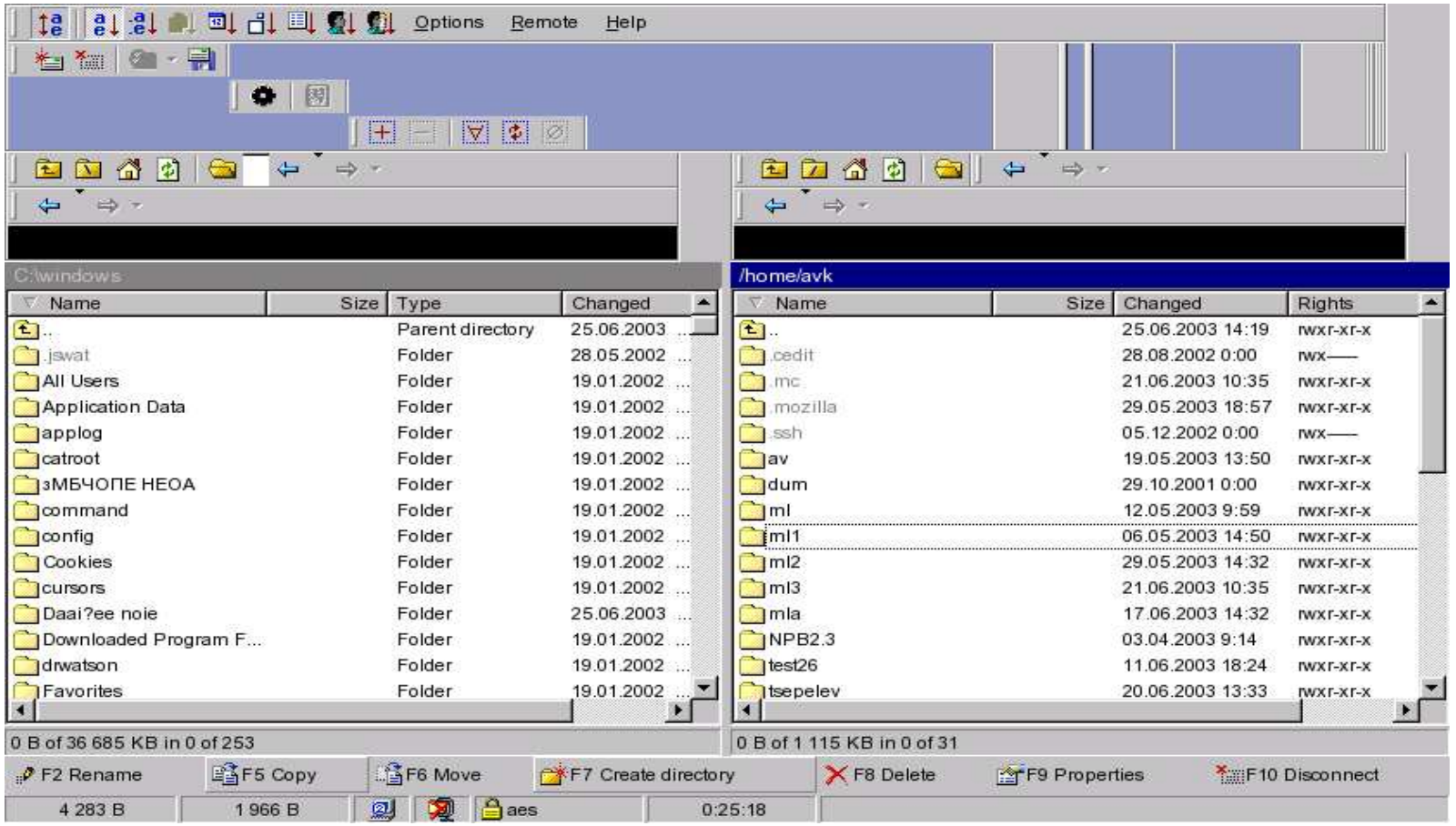
Современные процессоры для прикладных программистов

Valgrind

Замечательное средство отладки.
Поиск утечек, выходов за границы, мусоренья, if-ов по неинициализированным переменным и пр.

valgrind ./myprog

Winscp


Options
Remote
Help
C:\windows
Name
Size
Type
Changed
Parent directory
25.06.2003
Folder
28.05.2002
19.01.2002
25.06.2003
0 B of 36 685 KB in 0 of 253
/home/avk
Name
Size
Changed
Rights
25.06.2003 14:19
rwxr-xr-x
28.08.2002 0:00
rwx------
21.06.2003 10:35
29.05.2003 18:57
05.12.2002 0:00
19.05.2003 13:50
29.10.2001 0:00
12.05.2003 9:59
06.05.2003 14:50
29.05.2003 14:32
21.06.2003 10:35
17.06.2003 14:32
03.04.2003 9:14
11.06.2003 18:24
20.06.2003 13:33
0 B of 1 115 KB in 0 of 31
F2 Rename
F5 Copy
F6 Move
F7 Create directory
F8 Delete
F9 Properties
F10 Disconnect
4 283 B
1 966 B
aes
0:25:18

Основная проблема современных архитектур при решении счётных задач

Разрыв между пропускной способностью памяти и потребностью процессора в пропускной способности.

За время, когда CPU становится быстрее в 10 раз – память ускоряется только в 2! (При этом, ёмкость удесятеряется.)

2 способа замаскировать проблему

- Удлинить конвейер. (P4 – 20 стадий)

Не касается ни прикладных программистов, ни структуры современных ОС.
Ставит крест на любителях “легковесной многозадачности”.

- Кеш.

Длинный конвейер требует:

1. Спекулятивного исполнения.
2. Переименования регистров.
3. Предсказателя переходов.
4. Кеширования трассы.

Можно (возможно, наивно) надеяться, что обо всём подумает транслятор.

Впереди – 50-стадийные конвейеры.

Предсказатель переходов

0-ое поколение. Считаем, что переход всегда совершается. i860

1-ое. 1 или 2-битный счётчик. Пентиум просто.

2-е. Много счётчиков + история. Напр., P3: 16 2-х б счётчиков и 4 б трассы. 0101_2=5 – выбираем 5-ый счётчик. Легко решает проблему счётчиков по модулю 2.

Вычисление собственых значений. Изначальный вариант

```
     1 0.000%   3 0.000% :    DO 9 J=1,L
   0 0.000%     1 0.000% :    G=0.0
                  :C **********
  126 0.005%    2279 0.011% :    DO 6 K=1,J
521904 24.73%  5475404 27.58% :  6 G=G+A(J,K)*A(I,K)
 1294 0.061%   11206 0.056% :    JP1=J+1
   32 0.001%    1837 0.009% :    IF(L.LT.JP1)GO TO 8
   42 0.001%    5688 0.028% :    DO 7 K=JP1,L
520045 24.65%  4529287 22.81% :  7 G=G+A(K,J)*A(I,K)
                  :C **********
 1682 0.079%   10920 0.055% :  8 E(J)=G/H
   18 0.000%    1890 0.009% :    F=F+E(J)*A(I,J)
  691 0.032%    4904 0.024% :  9 CONTINUE
```

Вычисление собственых значений. После транспонирования

```
   15 0.000%     1 0.000% :   DO 9 J=1,L
   14 0.000%     0 0.000% :   G=0.0
                      :C **********
5444291 24.81% 28030574 23.50% :   DO 6 K=1,J
 15400 0.070%   78277 0.065% :  6 G=G+Atr(K,j)*Atr(K,i)
  3294 0.015%    5488 0.004% :   JP1=J+1
   676 0.003%     606 0.000% :   IF(L.LT.JP1)GO TO 8
7382918 33.64% 51024030 42.79% :   DO 7 K=JP1,L
 24077 0.109%  154796 0.129% :  7 G=G+Atr(J,k)*Atr(K,i)
                      :C **********
 16738 0.076%   39998 0.033% :  8 E(J)=G/H
  6034 0.027%   24976 0.020% :   F=F+E(J)*Atr(J,i)
  7003 0.031%   25883 0.021% :  9 CONTINUE
```

Собственные значения. Скорость для 2500*2500

	PIII/800	P4/2	α-21264 /600
Исходный	4m 49	7m 27	23m 11
Транспон.	2m 24	1m 6	7m 21
Кеш L1	16K/32B	8K/128B	64K/64B

Oprofile

“Неинтрузивное” профилирование на основе аппаратных счётчиков производительности. Графа вызовов нет.

Для PIII одновременно снимаются 2 показателя. Среди них:

- Время.
- Число команд.
- Число плавающих.
- Промахи в кеши.
- Число обращений к памяти.
- Задержки из-за нехватки ресурсов.
- Ошибки предсказателя переходов.

Социальный аспект.

МПЛЧ (см. сообщение М.И.Чащина)

Всё плохо:

- Короткие циклы,
- Загадочные if-ы

Без понимания, что же делается – ничего не ускоришь!

Задача Т.И.Серёжниковой. Изначальный вариант.

```
                :   do 42 kr1=kr1_min,kr1_max
    2 0.000%    1 0.000% :     do 62 kr2=2,127
                :       kr=128*(kr1-1)+kr2
                :
  719 0.007%  392 0.008% :       jo=1+(kr-0.5)/128
  492 0.005%    1 0.000% :       io=kr-(jo-1)*128+0.5
                :       sum2=0.
  239 0.002%    1 0.000% :       su2m(kr2,kr1)=0.
  216 0.002%    5 0.000% :       do 61 k=1,128
 6426 0.066% 2317 0.051% :         kr3=128*(k-1)+1
 1339 0.013%  183 0.004% :         y_cur=y(kr3)
20540 0.213% 6737 0.150% :         do 67 l=1,128
1062602 11.05% 146291 3.264% :       R=((sum(L,K)/128)/128-y_cur)
184102 1.915% 25516 0.569% :           R=R/(128*128)
319742 3.326% 118369 2.641% :           it=(129+(L-io))/2
                :           jt=(129+(K-jo))/2
2321112 24.14% 884519 19.73% :     su2m(kr2,kr1)=su2m(kr2,kr1)+
   R*(KER(IT,JT)/(128*128))
413984 4.306% 158979 3.547% :67         continue
 27926 0.290% 35539 0.792% :61       continue
```

После расщепления (18 % up)

```
   1 0.000%     0 0.000% :      do 100 kr2=2,127
 155 0.001%     7 0.000% :100      su2m(kr2,kr1)=0.
                :
   1 0.000%     7 0.000% :    do 200 k_sa=1,128,1
   4 0.000%     0 0.000% :      do 200 kr1=kr1_min,kr1_max
  29 0.000%     0 0.000% :        do 200 kr2=2,127
2243 0.023%     5 0.000% :          kr=128*(kr1-1)+kr2
110440 1.168%  63 0.003% :            jo=1+(kr-0.5)/128
57763 0.610%    4 0.000% :           io=kr-(jo-1)*128+0.5
               !c         do 200 k=1,128
62288 0.658%   21 0.001% :           do 200 k=k_sa,k_sa+0
44029 0.465%  44239 2.362% :             do 200 L=1,128
               :             kr3=128*(k-1)+l
904092 9.562%  8347 0.445% :           R=((sum(L,K)/128)/128-y(kr3))
296733 3.138%  2119 0.113% :              R=R/(128*128)
195481 2.067% 10744 0.573% :               it=(129+(L-io))/2
               :             jt=(129+(K-jo))/2
2208698 23.36% 23533 1.256% :       su2m(kr2,kr1)=su2m(kr2,kr1)
               : >       +R*(KER(IT,JT)/(128*128))
418672 4.428%  2679 0.143% :200  continue
```

Загадочный цикл

```
      do 20  k=1,128
        do 56 l=1,128
          sum1=0.
          kr=128*(k-1)+l
          do 21 j=1,JJ
            do 22 i=1,II
              JT=((k-J)+129)/2
              IT=((l-I)+129)/2
              sum1=sum1+x2d(i,j)*ker(it,jt)
22          continue
21        continue
          R=(sum1/128)/128-y(kr)
          sum2=sum2+R**2
56      continue
20    continue
```

a(i, t+1) = F(a(i-1,t), a(i,t), a(i+1,t)). Time-skewing (3T в кеше)

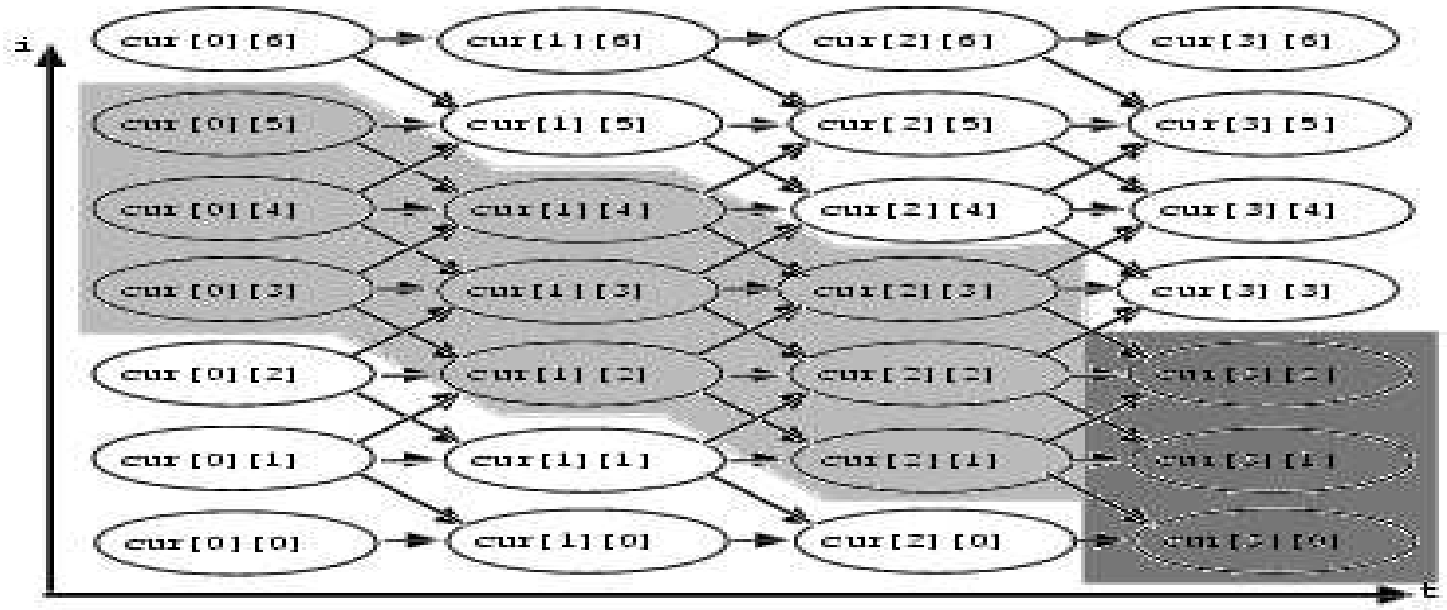


Figure 5: Dataflow Graph of Three Point Average Calculation

4 этапа

1. Где ? Для счётных задач обычно находится просто.
2. Есть ли резервы? Считаем количество команд (всех либо FP) за такт.
3. Почему? 2 базовые причины: конвейер и кеш. Oprofile поможет тоже.
4. Что можно сделать? Здесь собственно “творчество” и начинается.

Советы и Выводы

- Иногда думать о CPU полезно.
- Для научных вычислений главное – кеш.
- Oprofile вполне пригоден в жизни.
- Шаблонный способ ускорения – сделать максимум со строкой, уже попавшей в кеш.
- Компиляторы ничего не понимают. Им нужна помощь.

D.Bailey *12 способов одурачить массы, сообщая о производительности параллельных вычислений*

1) Приводите результаты для REAL, не для DOUBLE PRECISION.
2) Приводите результаты для “ядра приложения”, а не для всей задачи целиком.
3) Используйте низкоуровневые приёмы.
4) Увеличивайте размер задачи вместе с числом процессоров.
5) Экстраполируйте (линейно!) производительность на полномасштабную систему.
6) Сравнивайтесь со неоптимизированным скалярным кодом.
7) Сравнивайтесь со старыми машинами.

D.Bailey *12 способов одурачить массы, сообщая о производительности параллельных вычислений*

8) Сравнивайтесь с 1-процессорным вариантом параллельной задачи, не с последовательной задачей.

9) Сообщайте % загрузки процессоров, коэффициент ускорения (speed-up) или MFLOPS/$.

10) Изменяйте алгоритм под архитектуру. (Напр., Мегафлопсы хорошие, но сходится медленно)

11) Параллельные времена – в 1-задачном режиме, а последовательные – с многими пользователями.

12) Если ничего не помогает, не говорите о производительности вообще. Займитесь лучше визуализацией!