实验报告 2

1、

Hello World 的测试结果及个指标的含义：

```
sim_num_insn                 7942 # total number of instructions executed
sim_num_refs                 4337 # total number of loads and stores executed
sim_elapsed_time                1 # total simulation time in seconds
sim_inst_rate           7942.0000 # simulation speed (in insts/sec)
ld_text_base           0x00400000 # program text (code) segment base
ld_text_size                70128 # program text (code) size in bytes
ld_data_base           0x10000000 # program initialized data segment base
ld_data_size                 8192 # program init'ed `.data' and uninit'ed `.bs
s' size in bytes
ld_stack_base          0x7fffc000 # program stack segment base (highest addres
s in stack)
ld_stack_size               16384 # program initial stack size
ld_prog_entry          0x00400140 # program entry point (initial PC)
ld_environ_base        0x7fff8000 # program environment base address address
ld_target_big_endian            0 # target executable endian-ness, non-zero if
 big endian
mem.page_count                 26 # total number of pages allocated
mem.page_mem                 104k # total size of memory pages allocated
mem.ptab_misses                26 # total first level page table misses
mem.ptab_accesses          475648 # total page table accesses
mem.ptab_miss_rate         0.0001 # first level page table miss rate
```

每一行的前面是指标，中间的是对应的值，#后的是指标的含义。

bin/sslittle-na-sstrix-gcc hello.c 命令的含义：用 simplescalar 的编译器对 hello.c 进行编译，以生成能够在模拟器中运行的可执行文件，此条命令将 hello.c 编译成 a.out

simplesim-3.0/sim-safe a.out 命令的含义：用 sim-safe 对 a.out 进行模拟运行。

Sim-safe 是 simplescalar 中的一个模拟器，它会在指令的执行时检查指令的齐整性，检查访存指令的合法性等一些安全性检查。

sim_num_insn 7942 # total number of instructions executed

//执行的全部指令的条数

sim_num_refs 4337 # total number of loads and stores executed

//执行的装载和存储的指令条数

sim_elapsed_time 1 # total simulation time in seconds

//一秒内模拟次数

sim_inst_rate 7942.0000 # simulation speed (in insts/sec)

//

ld_text_base 0x00400000 # program text (code) segment base

//程序所在的代码段的基地址

ld_text_size 70128 # program text (code) size in bytes

//原程序大小,以 byte 为单位

ld_data_base 0x10000000 # program initialized data segment base

//程序初始化时的数据段地址

ld_data_size 8192 # program init'ed `.data' and uninit'ed `.bs s' size in bytes

ld_stack_base 0x7fff8000 # program stack segment base (highest addres s in stack)

//程序在堆栈段的基址

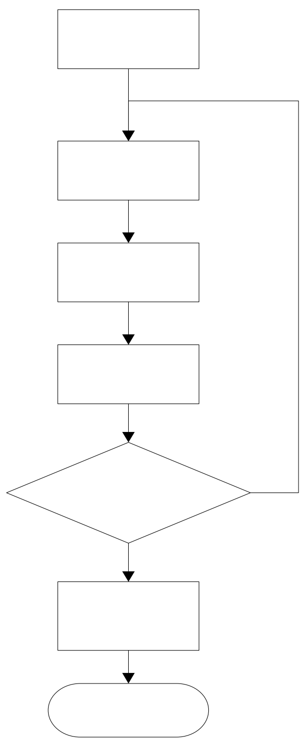
ld_stack_size 0 # program initial stack size
//程序所占用的堆栈段大小
ld_prog_entry 0x00400140 # program entry point (initial PC)
//程序入口地址（ 初始化指令寄存器 ）
ld_environ_base 0x7fff8000 # program environment base address address

ld_target_big_endian 0 # target executable endian-ness, non-zero if big endian

mem.page_count 26 # total number of pages allocated
//分配给程序的全部页数
mem.page_mem 104k # total size of memory pages allocated
//分配个程序的全部内存页的大小
mem.ptab_misses 26 # total first level page table misses
//未命中目录表总数
mem.ptab_accesses 475648 # total page table accesses
//访问的目录表总数
mem.ptab_miss_rate 0.0001 # first level page table miss rate
//页面失效率

交叉编译的概念：就是在一个平台上生成另一个平台上的可执行代码。这里需要注意的是所谓平台，实际上包含两个概念：体系结构（Architecture）、操作系统（Operating System）。同一个体系结
构可以运行不同的操作系统；同样，同一个操作系统也可以在不同的体系结构上运行。

2、SimpleScalar Simulator 模拟程序流程分析：



其中主要环节的过程如下：

1). 取指

sim-safe 调用 MD_FETCH_INST 宏，以程序计数器（以下简称 PC 寄存器）值为索引，在 sim-safe 模拟的内存空间中取出下一条指令。

2). 译码

sim-safe 调用 MD_SET_OPCODE 宏完成译码过程。sim-safe 曾在初始化时调用 md_init_decoder 函数初始化指令译码表，能够识别的指令都在这个指令译码表中有相应的下标。译码结束后，就能够得到该指令对应的数组下标，并能够获取这个数组下标内的 OP 值。

3). 模拟执行

负责模拟指令执行的文件(sim-safe.c)中对 DEFLINK、CONNECT 和 DEFINST 的定义如下：

```
#define DEFINST(OP,MSK,NAME,OPFORM,RES,FLAGS,O1,O2,O3,I1,I2,I3,I4)  \
        case OP:                                                       \
          SYMCAT(OP,_IMPL)                                           \
          break;
#define DEFLINK(OP,MSK,NAME,MASK,SHIFT)                              \
        case OP:                                                     \
          panic("attempted to execute a linking opcode");
#define CONNECT(OP)
#include "machine.def"
```

其中，DEFLINK、CONNECT、DEFINST 和 OP_IMPL 宏在指令模板 machine.def 文件中定义，这些宏的作用可以参看程序或 SimpleScalar 的文档。SYMCAT 宏的作用就是连接 OP 和_IMPL。 sim-safe 在译码过程结束后，已解析出取指阶段取得的指令对应的 OP 值，指令的模拟执行过程就是执行指令模板中对应该 OP 的 OP_IMPL 中的语句。如果 OP_IMPL 中的语句没有改变 PC 的值，则 PC 的值为程序顺序执行的下一地址；否则 PC 值为 OP_IMPL 语句中所赋的值。模拟执行结束后，即返回到取指过程。

应用程序结束时使用软件中断指令调用 exit 系统调用。sim-safe 模拟执行到 exit 系统调用，就结束当前应用程序的模拟，返回 sim-safe 的主流程。这时 sim-safe 输出应用程序动态执行的指令数、内存访问次数和模拟器执行时间等数据，并结束运行。

3、sim-outorder 的乱序流水线实现：

SimpleScalar 是使用 C 语言编写的，主流程从 main.c 文件中的主函数 main()开始。阅读 main()可以看到它主要在作模拟前的准备工作，而真正的模拟执行跳转到 sim-outorder.c 文件中模拟主函数 sim_main()。函数 sim_main()主干是一个死循环，它的流程完全模拟流水线的处理过程，不同的是由于模拟的需要将流水的顺序逆转，以使逻辑上的流水站不要提前处理要下一拍才能处理的指令。模拟结束通过信号量返回主函数 main()，退出。

看比较直观的模拟主函数sim_main()流程：

首先是一些准备工作，然后开始进入 5 级流水：

ruu_commit()→ruu_writeback()→ruu_issue()→ruu_dispatch()→ruu_fetch()，然后在每个流水周期结束后，cycle 需要加“1”。

SimpleScalar 是采用的执行方式模拟，所以在模拟器中要对目标处理器＋存储器的几乎全部单元进行说明，而被模拟程序就在这些虚拟的部件上“运行”，从中得到统计分析所

需的数据。

4、 以 Alpha 为例，用 sim-cache 模拟运行 tests-alpha 目录下的程序。

使用 sim-cache 模拟程序的格式如下:

/sim-cache -cache:dl1 dl1:<nsets>:<bsize>:<assoc>:<rpolicy> ./**.out (**.out 为要测试文件)

其中各参数的的含义如下:

<nsets> is the number of sets in the structure, it must be a positive integer and a power of two.

<bsize> is the block size(块的大小), it also must be a positive integer and a power of two.

<assoc> is the associativity(相联度), which must be a positive integer.

<rpolicy> is the replacement policy and is either l (LRU), r (random) or f (FIFO).(替换算法)

在 simplescalar 目录下对 simplesim-3.0/tests-alpha/src/test-math.c 进行交叉编译,命令为:

[root@localhost simplescalar]# bin/sslittle-na-sstrix- gcc ./simplesim-3.0/tests-alpha/src/test-math.c

[root@localhost simplescalar]# simplesim-3.0/sim-cache -cache:dl1 dl1:1024:32:1:l ./a.out

得到运行结果:

```
sim: ** starting functional simulation w/ caches **
pow(12.0, 2.0) == 144.000000
pow(10.0, 3.0) == 1000.000000
pow(10.0, -3.0) == 0.001000
str: 123.456
x: 123.000000
str: 123.456
x: 123.456000
str: 123.456
x: 123.456000
123.456 123.456000 123 1000
sinh(2.0) = 3.62686
sinh(3.0) = 10.01787
h=3.60555
atan2(3,2) = 0.98279
pow(3.60555,4.0) = 169
169 / exp(0.98279 * 5) = 1.24102
3.93117 + 5*log(3.60555) = 10.34355
cos(10.34355) = -0.6068,   sin(10.34355) = -0.79486
x     0.5x
x0.5     x
x   0.5x
-1e-17 != -1e-17 Worked!
```

得到 cache 的各项参数(部分)如下 :

```
mem.page_count                 33 # total number of pages allocated
mem.page_mem                 132k # total size of memory pages allocated
mem.ptab_misses                34 # total first level page table misses
```

mem.ptab_accesses 1545425 # total page table accesses
mem.ptab_miss_rate 0.0000 # first level page table miss rate

修改参数<assoc> 32->1024,
[root@localhost simplescalar]# simplesim-3.0/sim-cache -cache:dl1 dl1:1024:1024:1:l ./a.out
得到新的数据:
mem.page_count 27 # total number of pages allocated
mem.page_mem 108k # total size of memory pages allocated
mem.ptab_misses 27 # total first level page table misses
mem.ptab_accesses 575798 # total page table accesses
mem.ptab_miss_rate 0.0000 # first level page table miss rate
当 cache 块增大时所需页的总数及内存页的大小相应减少.

修改替换算法<rpolicy>l->f
[root@localhost simplescalar]# simplesim-3.0/sim-cache -cache:dl1 dl1:1024:32:1:f ./a.out
mem.page_count 33 # total number of pages allocated
mem.page_mem 132k # total size of memory pages allocated
mem.ptab_misses 34 # total first level page table misses
mem.ptab_accesses 1545425 # total page table accesses
mem.ptab_miss_rate 0.0000 # first level page table miss rate
由于页面失效率为 0 所以修改替换算法对 cache 参数没有影响.

自己设计 4 个 c 程序如下:
(1):
```
#include<stdio.h>
int s[20000]={0};
void self_numbers(){
    int i,ii,j,k,kk,sum;
    for(i=1;i<=10000;i++){
      sum=i;
      j=1;
      k=10;
      while((i/k)>0){k=k*10;j++;}
      k=10;
      kk=1;
      for(ii=1;ii<=j;ii++){sum+=(i%k)/kk;k*=10;kk*=10;}
      s[sum]++;
      }
    for(i=1;i<10000;i++)if(s[i]==0)printf("%d\n",i);
}
int main(){
    self_numbers();
    return 0;
}
```

交叉编译如下:

[root@localhost simplescalar]# bin/sslittle-na-sstrix-gcc ./simplesim-3.0/tests-alpha/src/test-plus.c

[root@localhost simplescalar]# simplesim-3.0/sim-cache -cache:dl1 dl1:1024:32:1:f ./a.out

得到如下参数:

```
mem.page_count                 46 # total number of pages allocated
mem.page_mem                 184k # total size of memory pages allocated
mem.ptab_misses                48 # total first level page table misses
mem.ptab_accesses        20705192 # total page table accesses
mem.ptab_miss_rate         0.0000 # first level page table miss rate
```

修改:dl1:1024:32:1:f -> dl1:1024:32:1:l

得到如下参数:

```
mem.page_count                 46 # total number of pages allocated
mem.page_mem                 184k # total size of memory pages allocated
mem.ptab_misses                48 # total first level page table misses
mem.ptab_accesses        20705192 # total page table accesses
mem.ptab_miss_rate         0.0000 # first level page table miss rate
```

此时修改替换算法没有影响.

修改:dl1:1024:32:1:f -> 修改:dl1:1024:4096:1:f

得到如下参数:

```
mem.page_count                 40 # total number of pages allocated
mem.page_mem                 160k # total size of memory pages allocated
mem.ptab_misses                40 # total first level page table misses
mem.ptab_accesses          602134 # total page table accesses
mem.ptab_miss_rate         0.0001 # first level page table miss rate
```

块增大总页数减少,但缺页率增大.

修改:dl1:1024:32:1:f -> 修改:dl1:1024:4096:1:l

得到如下参数:

```
mem.page_count                 40 # total number of pages allocated
mem.page_mem                 160k # total size of memory pages allocated
mem.ptab_misses                40 # total first level page table misses
mem.ptab_accesses          602134 # total page table accesses
mem.ptab_miss_rate         0.0001 # first level page table miss rate
```

修改替换算法没有影响.

(2):

```
#define maxn 2001
#include<stdio.h>
#include<string.h>
int g[3]={0,2,1};
```

```
int num,fro,beh,n,q,m,x,y,p[2*maxn],f[maxn][maxn],use[maxn],color[maxn];
void task1()
{ int j1;
   memset(f,0,sizeof(f));memset(use,0,sizeof(use));memset(color,0,sizeof(color));
   scanf("%d %d",&n,&m);
   for(j1=1;j1<=m;j1++)
    {scanf("%d %d",&x,&y);
     f[x][0]+=1;f[x][f[x][0]]=y;
     f[y][0]+=1;f[y][f[y][0]]=x;
    }
 }
int main()
{int i1,j,z;
 scanf("%d",&q);
 for (i1=1;i1<=q;i1++)
   { task1();z=0;num=0;
     do
      {for (j=1;j<=n;j++) if (use[j]==0) break;
      memset(p,0,sizeof(p));use[j]=1;color[j]=1;p[1]=j;fro=0;beh=1;num+=1;
      do
       {fro+=1;
       for(j=1;j<=f[p[fro]][0];j++)
        if (use[f[p[fro]][j]]==1&&color[f[p[fro]][j]]==color[p[fro]]) {z=1;break;}
           else if (use[f[p[fro]][j]]==0)
                  {
                   beh+=1;p[beh]=f[p[fro]][j];
                   color[f[p[fro]][j]]=g[color[p[fro]]];
                   use[f[p[fro]][j]]=1;num+=1;
                  }
       if (z==1) break;}
      while (fro<=beh);
       if (z==1) break; }
    while (num<n);
    printf("Scenario #%d:\n",i1);
    if(z==1) printf("Suspicious bugs found!\n");
    else printf("No suspicious bugs found!\n");
    printf("\n");
   }
return 0;
}
```

交叉编译如下:

```
[root@localhost simplescalar]# bin/sslittle-na-sstrix-gcc ./simplesim-3.0/tests-alpha/src/test-2.c
[root@localhost simplescalar]# simplesim-3.0/sim-cache -cache:dl1 dl1:1024:32:1:f ./a.out
```

得到如下参数:

```
mem.page_count              3951 # total number of pages allocated
mem.page_mem              15804k # total size of memory pages allocated
mem.ptab_misses             8101 # total first level page table misses
mem.ptab_accesses       64949048 # total page table accesses
mem.ptab_miss_rate        0.0001 # first level page table miss rate
```

修改:dl1:1024:32:1:f -> dl1:1024:32:1:l

得到如下参数:

```
mem.page_count              3951 # total number of pages allocated
mem.page_mem              15804k # total size of memory pages allocated
mem.ptab_misses             8101 # total first level page table misses
mem.ptab_accesses       64949048 # total page table accesses
mem.ptab_miss_rate        0.0001 # first level page table miss rate
```

修改替换算法对页面的使用没有影响.

修改:dl1:1024:32:1:f -> dl1:1024:4096:1:f

得到如下参数:

```
mem.page_count                28 # total number of pages allocated
mem.page_mem                112k # total size of memory pages allocated
mem.ptab_misses               28 # total first level page table misses
mem.ptab_accesses         609526 # total page table accesses
mem.ptab_miss_rate        0.0000 # first level page table miss rate
```

当页面增大时页面失效率降低.

修改:dl1:1024:32:1:f -> dl1:8:32:1:f

得到如下参数:

```
mem.page_count              3951 # total number of pages allocated
mem.page_mem              15804k # total size of memory pages allocated
mem.ptab_misses             8101 # total first level page table misses
mem.ptab_accesses       64949048 # total page table accesses
mem.ptab_miss_rate        0.0001 # first level page table miss rate
```

降低参数<nsets>对页面使用无影响.

修改:dl1:1024:32:1:f -> dl1:1024:32:1024:f

得到如下参数:

```
mem.page_count              3951 # total number of pages allocated
mem.page_mem              15804k # total size of memory pages allocated
mem.ptab_misses             8101 # total first level page table misses
mem.ptab_accesses       64949048 # total page table accesses
mem.ptab_miss_rate        0.0001 # first level page table miss rate
```

增大<assoc> (相联度)对页面使用无影响.

(3):

```
#include <math.h>
#include <stdio.h>
#include <string.h>
#define Max(x,y) ((x)>(y)?(x):(y))
#define Min(x,y) ((x)<(y)?(x):(y))
#define MaxN 100100
#define MaxH 18
#define INF 1000000007
int v[MaxH][MaxN];
int H, N, M;
int _s[MaxH*2], _e[MaxH*2], _l[MaxH*2], top;
int getH(int x){
    int i,t,res;
    x--;
    res=0;
    t=1;
    for (i=0;i<32;i++){   if (x&t) res=i+1;   t<<=1; }
    return res;
}
void MergeSort(int s, int t, int h){
    int m,i,j,k;
    if (h==H) return;
    m=(s+t+1)/2;
    MergeSort(s,m,h+1);
    MergeSort(m,t,h+1);
    for (i=s,j=m,k=s;i<m&&j<t;k++){
        if (v[h+1][i]<v[h+1][j]) v[h][k]=v[h+1][i++];
        else v[h][k]=v[h+1][j++];
        } while (i<m) v[h][k++]=v[h+1][i++];
    while (j<t) v[h][k++]=v[h+1][j++];
}
void getSegment(int s, int e, int rs, int re, int l){
    int m;
    if (re<=s || rs>=e) return;
    if (rs<=s && re>=e){
        _s[top]=s;_e[top]=e;_l[top]=l;top++;   return;
        }
    if (l==H) return;
    m=(s+e+1)/2;
    getSegment(s,m,rs,re,l+1);
    getSegment(m,e,rs,re,l+1);
}
int mybsearch(int val, int a[], int L){
```

```
    int low,hig,mid; low=0;hig=L;
    while (low<hig){
        mid=(low+hig)/2;
        if (a[mid]==val)
        return mid+INF;
        if (a[mid]>val) hig=mid;
        else low=mid+1;
        }
        return hig;
}
void find(int s, int e, int k){
    int i,min,max,mid,res,resk;
    char found; top=0;
    getSegment(0,N,s,e,0);
    for (i=0,min=INF,max=-INF;i<top;i++){
          min=Min(min,v[_l[i]][_s[i]]);
        max=Max(max,v[_l[i]][_e[i]-1]);
    }
    while (min<=max){
        found=0;
        res=0;
        mid=(min+max)/2;
        for (i=0;i<top;i++){
            resk=mybsearch(mid,v[_l[i]]+_s[i],_e[i]-_s[i]);
            if (resk>=INF){
                found=1;
                resk-=INF;
            }
            res+=resk;
        }
        if (found && res==k){
            printf("%d\n", mid);
            return;
        }
        if (res<=k) min=mid+1;
        else max=mid-1;
    }
}
void solve(){
    int i,s,e,k; H=getH(N);
    for (i=0;i<N;i++)
        scanf("%d", &v[H][i]);
    MergeSort(0,N,0);
    while (M--){
```

```
        scanf("%d %d %d", &s, &e, &k);
        s--;
        k--;
        find(s,e,k);
    }
}
int main(){
    scanf("%d %d", &N, &M);
    solve();
    return 0;
}
```

交叉编译如下:

[root@localhost simplescalar]# bin/sslittle-na-sstrix-gcc ./simplesim-3.0/tests-alpha/src/test-3.c
[root@localhost simplescalar]# simplesim-3.0/sim-cache -cache:dl1 dl1:1024:32:1:f ./a.out

得到如下参数:

```
mem.page_count                 38 # total number of pages allocated
mem.page_mem                 152k # total size of memory pages allocated
mem.ptab_misses                40 # total first level page table misses
mem.ptab_accesses          734872 # total page table accesses
mem.ptab_miss_rate         0.0001 # first level page table miss rate
```

修改:dl1:1024:32:1:f -> dl1:1024:32:1:l
得到如下参数:

```
mem.page_count                 38 # total number of pages allocated
mem.page_mem                 152k # total size of memory pages allocated
mem.ptab_misses                40 # total first level page table misses
mem.ptab_accesses          734872 # total page table accesses
mem.ptab_miss_rate         0.0001 # first level page table miss rate
```

修改替换算法对页面的使用没有影响.

修改:dl1:1024:32:1:f -> dl1:1024:1024:1:f
得到如下参数:

```
mem.page_count                 29 # total number of pages allocated
mem.page_mem                 116k # total size of memory pages allocated
mem.ptab_misses                29 # total first level page table misses
mem.ptab_accesses          619830 # total page table accesses
mem.ptab_miss_rate         0.0000 # first level page table miss rate
```

当页面增大时页面失效率降低.

修改:dl1:1024:32:1:f -> dl1:8:32:1:f
得到如下参数:

```
mem.page_count                 38 # total number of pages allocated
```

mem.page_mem 152k # total size of memory pages allocated
mem.ptab_misses 40 # total first level page table misses
mem.ptab_accesses 734872 # total page table accesses
mem.ptab_miss_rate 0.0001 # first level page table miss rate
降低参数<nsets>对页面使用无影响.

修改:dl1:1024:32:1:f -> dl1:1024:32:1024:f
得到如下参数:
mem.page_count 38 # total number of pages allocated
mem.page_mem 152k # total size of memory pages allocated
mem.ptab_misses 40 # total first level page table misses
mem.ptab_accesses 734872 # total page table accesses
mem.ptab_miss_rate 0.0001 # first level page table miss rate
增大<assoc> (相联度)对页面使用无影响.

(4):
```
#include <math.h>
#include <ctype.h>
#include <stdio.h>
#include <string.h>
double X0 , Y0 , R ;
double point[200][2];
int pN ;
int zaiyou ( int j , int i ) ;
int zaizuo ( int j , int i ) ;
void i_Done ()
{
    int Res = 0 , i , j , bb;
    for ( i = 0 ;i < pN ;i ++ ){
        bb = 0 ,j;
        for ( j = 0 ;j <pN ; j++ ) {
            if( zaiyou ( j,i ) )
                bb ++ ;
        }
        if ( bb*2 <pN ) bb= pN-bb ;
        if ( Res < bb )
            Res = bb ;
        bb = 0 ;
        for ( j = 0 ;j <pN ; j++ ) {
            if( zaizuo ( j,i ) )
                bb ++ ;
        }
        if ( bb*2 <pN ) bb= pN-bb ;
        if ( Res < bb )
```

```
            Res = bb ;
    }
      printf("%d\n",Res);
}

int zaiyou ( int j , int i )
{
     double x0= X0 , y0 = Y0 ,y2y;
     double x1=point[i][0] ,y1=point[i][1] ;
     double x2=point[j][0] ,y2=point[j][1] ;
     if ( x0 == x1 && y0==y1 ) return 0 ;
     if ( x0==x1 ) return ( x2 >= x1 ) ;
     if ( y0==y1 ) return ( y2 >= y1 ) ;
     y2y= (x2-x0)*(y1-y0)+y0*(x1-x0) ;
     return ( y2y >= y2*(x1-x0) ) ;
}

int zaizuo ( int j , int i )
{
     double x0= X0 , y0 = Y0 ,y2y;
     double x1=point[i][0] ,y1=point[i][1] ;
     double x2=point[j][0] ,y2=point[j][1] ;
     if ( x0 == x1 && y0==y1 ) return 0 ;
     if ( x0==x1 ) return ( x2 <= x1 ) ;
     if ( y0==y1 ) return ( y2 <= y1 ) ;
     y2y= (x2-x0)*(y1-y0)+y0*(x1-x0) ;
     return ( y2y <= y2*(x1-x0) ) ;
}



int main()
{
     int N,i;
       double x , y ;
       scanf("%lf%lf%lf",&X0,&Y0,&R);
     if( R>=0 ) {
              scanf("%d",&N);
          pN = 0 ;
          memset ( point, 0 ,sizeof ( point ) ) ;
          for ( i = 0 ;i < N ;i ++ ) {

               scanf("%lf%lf",&x,&y);
               if ( pow(X0-x,2) + pow(Y0-y,2) <= R*R ) {
```

```
                point[pN][0] = x ;
                point[pN][1] = y ;
                pN ++ ;
            }
        }
        i_Done () ;
    }
    return 0 ;
}
```

交叉编译如下:

[root@localhost simplescalar]# bin/sslittle-na-sstrix-gcc ./simplesim-3.0/tests-alpha/src/test-3.c

[root@localhost simplescalar]# simplesim-3.0/sim-cache -cache:dl1 dl1:1024:32:1:f ./a.out

得到如下参数:

```
mem.page_count              34 # total number of pages allocated
mem.page_mem              136k # total size of memory pages allocated
mem.ptab_misses             36 # total first level page table misses
mem.ptab_accesses       718382 # total page table accesses
mem.ptab_miss_rate      0.0001 # first level page table miss rate
```

修改:dl1:1024:32:1:f -> dl1:1024:32:1:l

得到如下参数:

```
mem.page_count              34 # total number of pages allocated
mem.page_mem              136k # total size of memory pages allocated
mem.ptab_misses             36 # total first level page table misses
mem.ptab_accesses       718382 # total page table accesses
mem.ptab_miss_rate      0.0001 # first level page table miss rate
```

修改替换算法对页面的使用没有影响.

修改:dl1:1024:32:1:f -> dl1:1024:8:1:f

得到如下参数:

```
mem.page_count              34 # total number of pages allocated
mem.page_mem              136k # total size of memory pages allocated
mem.ptab_misses             36 # total first level page table misses
mem.ptab_accesses       718382 # total page table accesses
mem.ptab_miss_rate      0.0001 # first level page table miss rate
```

无影响.

修改:dl1:1024:32:1:f -> dl1:8:32:1:f

得到如下参数:

```
mem.page_count              34 # total number of pages allocated
mem.page_mem              136k # total size of memory pages allocated
mem.ptab_misses             36 # total first level page table misses
```

mem.ptab_accesses 718382 # total page table accesses

mem.ptab_miss_rate 0.0001 # first level page table miss rate

降低参数<nsets>对页面使用无影响.

修改:dl1:1024:32:1:f -> dl1:1024:32:1024:f

得到如下参数:

mem.page_count 34 # total number of pages allocated

mem.page_mem 136k # total size of memory pages allocated

mem.ptab_misses 36 # total first level page table misses

mem.ptab_accesses 718382 # total page table accesses

mem.ptab_miss_rate 0.0001 # first level page table miss rate

增大<assoc> (相联度)对页面使用无影响.