Prof. Charles E. Leiserson
Prof. Michael Bender
Prof. Hsu Wen Jing
Dr. Bradley C. Kuszmaul

Massachusetts Institute of Technology
6.895: Theory of Parallel Systems
Handout 3
September 10, 2003

Code for Sort Algorithms

C Code for Insertion Sort

```
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <sys/resource.h>
#include <unistd.h>
#include <string.h>

/* Insertion sort */
void isort (int *x, int n) {
    int i,j;
    for (i=1; i<n; i++) {
        int nv=x[i];
        /* everything before x[i] is sorted. */
        for (j=0; j<i; j++) {
            int xj=x[j];
            if (xj>nv) {
                x[j]=nv;
                nv=xj;
            }
        }
        x[j]=nv;
    }
}
```

C Code for Insertion Sort

C Code for Bitonic Sort of 5 Values

```
/* A special case sorting function that sorts 5 values.
 * Load the values into variables, swap them around, and store them out */
void sort5 (int *x) {
    #define MSWAP(x,y) if (x>y) { tmp=x; x=y; y=tmp; }
    int a0=x[0];
    int a1=x[1];
    int a2=x[2];
    int a3=x[3];
    int a4=x[4];
    int tmp;
    /* This is a minimal sorting network for five values. */
    MSWAP(a1,a2);
    MSWAP(a3,a4);
    MSWAP(a1,a3);
    MSWAP(a0,a2);
    MSWAP(a2,a4);
    MSWAP(a0,a3);
    MSWAP(a0,a1);
    MSWAP(a2,a3);
    MSWAP(a1,a2);
    //assert((a0<a1) && (a1<a2) && (a2<a3) && (a3<a4));
    x[0]=a0;
    x[1]=a1;
    x[2]=a2;
    x[3]=a3;
    x[4]=a4;
}
```

C Code for Bitonic Sort of 5 Values

C Code for Testing Sorts

```
/* The rest of this file constructs random data, and runs sorts many
 * times, and measures the time */

double rdiff (struct rusage *rstart, struct rusage *rend) {
    return rend->ru_utime.tv_sec - rstart->ru_utime.tv_sec
        + (rend->ru_utime.tv_usec-rstart->ru_utime.tv_usec)*1e-6;
}

enum { N = 5, DR=100, TRIALS=10000000 };
int main (int argc, char *argv[]) {
    struct rusage rstart, rend;
    int V[DR*N], X[N];
    int i;
    int off=0;
    double callibrate;
    int two;
    assert(argc==1);
    for (i=0; i<DR*N; i++) V[i]=random();
    // Do the callibration twice
    for (two=0; two<2; two++) {
        getrusage(RUSAGE_SELF, &rstart);
        for (i=0; i<TRIALS; i++) {
            memcpy(X, V+off, sizeof(int)*N);
            off+=N; if (off>=DR*N) off=0;
        }
        getrusage(RUSAGE_SELF, &rend);
    }
    callibrate=rdiff(&rstart, &rend);

    for (two=0; two<2; two++) {
        getrusage(RUSAGE_SELF, &rstart);
        for (i=0; i<TRIALS; i++) {
            memcpy(X, V+off, sizeof(int)*N);
            off+=N; if (off>=DR*N) off=0;
            isort(X,N);
        }
        getrusage(RUSAGE_SELF, &rend);
    }
    printf("%s isort(%d) time %f\n", argv[0], N, rdiff(&rstart, &rend)-callibrate);

    assert(N==5);
    for (two=0; two<2; two++) {
        getrusage(RUSAGE_SELF, &rstart);
        for (i=0; i<TRIALS; i++) {
            memcpy(X, V+off, sizeof(int)*N);
            off+=N; if (off>=DR*N) off=0;
            sort5(X);
        }
        getrusage(RUSAGE_SELF, &rend);
    }
    printf("%s sort5 time %f\n", argv[0], rdiff(&rstart, &rend)-callibrate);

    return 0;
}
```

C Code for Testing Sorts

Excerpt of Assembly Code for Bitonic Sort

```
sort5:  # 0xe0
        .dynsym sort5   sto_default
        .frame  $sp, 0, $31
        .loc    1 28 21
 #  26
 #  27
 #  28  void sort5 (int *x) {
.BB1.sort5:     # 0xe0
 #<freq>
 #<freq> BB:1 frequency = 1.00000 (heuristic)
 #<freq>
        .loc    1 34 9
 #  30      int a0=x[0];
 #  31      int a1=x[1];
 #  32      int a2=x[2];
 #  33      int a3=x[3];
 #  34      int a4=x[4];
        lw $5,16($4)                    # [0]  id:110
        .loc    1 33 9
        lw $24,12($4)                   # [1]  id:109
        .loc    1 31 9
        lw $14,4($4)                    # [2]  id:107
        .loc    1 32 9
        lw $13,8($4)                    # [3]  id:108
        .loc    1 38 5
 #  35      int tmp;
 #  36      /* Do a minimal sort */
 #  37      MSWAP(a1,a2);
 #  38      MSWAP(a3,a4);
        or $8,$24,$0                    # [3]
        .loc    1 37 5
        slt $15,$5,$24                  # [3]
        or $11,$14,$0                   # [4]
        movz $8,$5,$15                  # [4]
        .loc    1 34 9
        slt $12,$13,$14                 # [5]
        movz $5,$24,$15                 # [5]
        movz $11,$13,$12                # [6]
        .loc    1 30 9
        lw $7,0($4)                     # [7]  id:106
        movz $13,$14,$12                # [7]
        .loc    1 39 5
 #  39      MSWAP(a1,a3);
```

```
        or $3,$13,$0                    # [8]
        .loc    1 38 5
        slt $12,$5,$13                  # [8]
        .loc    1 40 5
 #  40      MSWAP(a0,a2);
        or $25,$7,$0                    # [9]
        movz $3,$5,$12                  # [9]
        movz $5,$13,$12                 # [10]
        .loc    1 39 5
        slt $6,$11,$7                   # [10]
        movz $25,$11,$6                 # [11]
        movz $11,$7,$6                  # [12]
        .loc    1 42 5
 #  41      MSWAP(a2,a4);
 #  42      MSWAP(a0,a3);
        or $1,$11,$0                    # [13]
        .loc    1 41 5
        slt $6,$3,$11                   # [13]
        or $10,$25,$0                   # [14]
        movz $1,$3,$6                   # [14]
```

Exceprt of Assembly Code for Bitonic Sort