Tentamen Parallel Computing

Maandag 24 augustus 2009, 14:00-17:00 uur

docenten: R. de Bruin, A. Meijster

- Lees eerst een opgave volledig door alvorens deze te maken.
- Geef geen antwoorden zonder toelichting.
- Het tentamen bestaat uit vier opgaven. Alle opgaven zijn evenveel punten waard.
- Succes.

Opgave 1: Architectuur

(a) Computer architecturen worden volgens de taxonomie van Flynn ingedeeld in vier klasses, namelijk SISD, SIMD, MIMD en MISD. Geef van iedere klasse aan waar de afkorting voor staat. Geef voor de eerste drie klasses (SISD, SIMD, en MIMD) een voorbeeld van een architectuur uit de betreffende klasse. Waarom bestaat er geen voorbeeld voor de klasse MISD?

(b) Wat is het verschil tussen een scalaire processor en een vectorprocessor?

(c) Beschouw een vectorprocessor met vectorregisters ter lengte 8. De onderstaande vier programmafragmenten zullen op deze processor niet erg efficient draaien (veronderstel dat de compiler geen optimalisaties uitvoert!). Geef van ieder fragment aan wat het probleem is en hoe dit kan worden opgelost.

```
/* code fragment 1 */
double a[1024], b[1024], c[1024];
double s;
int i;
for (i=0; i<1024; i++) {
  s = 2*a[i];
  c[i] = s + b[i];
}
```

```
/* code fragment 2 */
double a[1024], b[4][1024];
int i, j;
for (i=0; i<1024; i++) {
  for (j=0; j<4; j++) {
    a[i] = a[i] + b[j][i];
  }
}
```

```
/* code fragment 3 */
double a[1024], b[1024], c[1024];
int i;
for (i=0; i<1024; i++) {
  a[i] = sqrt(b[i]);
  c[i] = 2*a[i];
}
```

```
/* code fragment 4 */
double a[1024], b[1024], c[1024];
int i;
for (i=1; i<1024; i++) {
  a[i-1] = b[i];
  c[i] = a[i];
}
```

(d) Vrijwel alle SMP machines met scalaire processoren beschikken over zgn. *write-back caches*. Deze caches bestaan uit meerdere cache-lines, waarbij iedere cache-line zich in één van de volgende toestanden kan bevinden: *clean*, *dirty*, *invalid* of *shared*. Leg uit wat deze toestanden

betekenen en geef een voorbeeld. Leg ook uit wat *false-sharing* is.

(e) Een programma bestaat uit een deel dat parallel uitgevoerd kan worden, en een deel dat inherent sequentieel is. Laat S de fractie van de executietijd zijn die inherent sequentieel is (S is een fractie, dus $0 \leq S \leq 1$). Laat zien dat de theoretische maximale speedup die bereikt kan worden met P processoren gegeven wordt door:

$$S(P) = \frac{1}{S + \frac{1-S}{P}}.$$

Hoe heet de bovenstaande wet? Heeft het zin om een supercomputer te kopen met 1024 processoren voor een probleem waarvan de fractie sequentiële tijd 1% is?

Opgave 2: OpenMP
Het onderstaande programma bestaat uit drie threads. De threads delen de *shared variable* `x`.

```
int main (int argc, char **argv)
{
  int x = 1;
  #pragma omp parallel sections shared(x)
  {
    #pragma omp section
    {
      #pragma omp atomic
      x = x + 1;
    }
    #pragma omp section
    {
      #pragma omp atomic
      x = 2*(x+1);
    }
    #pragma omp section
    {
      #pragma omp atomic
      x = 2*(x-1);
    }
  }
  printf ("x=%d\n", x);
  return EXIT_SUCCESS;
}
```

(a) Eén van de mogelijke uitkomsten is `x=3`. Welke uitkomsten zijn er nog meer mogelijk? Geef voor iedere uitkomst aan in welke volgorde de toekenningen zijn uitgevoerd.

We veranderen het programma nu in het volgende programma

```
int main (int argc, char **argv)
{
  int x = 1;
  #pragma omp parallel sections shared(x)
  {
    #pragma omp section
    {
      #pragma omp atomic
      x = x + 1;
    }
    #pragma omp section
    {
      #pragma omp atomic
      int y = 2*(x+1);
      #pragma omp atomic
      x = y;
    }
    #pragma omp section
    {
      #pragma omp atomic
      int y = 2*(x-1);
      #pragma omp atomic
      x = y;
    }
  }
  printf ("x=%d\n", x);
  return EXIT_SUCCESS;
}
```

(b) Zijn er nu meer uitkomsten mogelijk (dan in onderdeel a)? Zo ja, geef een voorbeeld. Zo nee, waarom niet?

(c) Het onderstaande programmafragment implementeert de transpositie van een matrix `A`.

```
double tmp, A[1024][1024];
int i, j;
for (i=0; i<1024; i++)
{
  for (j=i+1; j<1024; j++)
  {
    tmp = A[i][j];
    A[i][j] = A[j][i];
    A[j][i] = tmp;
  }
}
```

Geef aan hoe je dit fragment m.b.v. OpenMP kan parallelliseren. Houd rekening met load-balancing!

(d) Geef minstens twee redenen waarom proces-synchronisatie (bijvoorbeeld het implementeren van een semafoor) met behulp van OpenMP niet mogelijk is.

Opgave 3: Pthreads
Het onderstaande programmafragment implementeert een semafoor.

```
 1 typedef struct Semstruct {
 2     pthread_mutex_t mutex;
 3     pthread_cond_t  cond;
 4     unsigned int    value;
 5 } Semaphore;
 6
 7 void initSemaphore(Semaphore *s, unsigned int v) {
 8    /* this routine initializes the semaphore s by setting
 9     * its value to v. You are not requested to implement this.
10     */
11 }
12
13 void downSemaphore(Semaphore *s) {
14     pthread_mutex_lock(&(s->mutex));
15     while (s->value == 0) {
16       pthread_cond_wait(&(s->cond), &(s->mutex));
17     }
18     s->value--;
19     pthread_mutex_unlock(&(s->mutex));
20 }
21
22 void upSemaphore(Semaphore *s) {
23     pthread_mutex_lock(&(s->mutex));
24     s->value++;
25     if (s->value == 1) {
26       pthread_cond_signal(&(s->cond));
27     }
28     pthread_mutex_unlock(&(s->mutex));
29 }
```

(a) Wat is het essentiële verschil tussen een semafoor en een mutex?

(b) Waarom wordt in de regels 15–17 gebruik gemaakt van een `while`-lus in plaats van een eenvoudig `if`-statement?

(c) Waarom wordt in regel 26 gebruik gemaakt van `pthread_cond_signal` en niet van `pthread_cond_broadcast`?

(d) Schrijf (pseudo-)code voor twee threads die vijfmaal om de beurt respectievelijk `ping` en `pong` afdrukken waarbij de synchronisatie plaats dient te vinden d.m.v. de bovenstaande code voor semaforen.

(e) Schrijf (pseudo-)code voor een routine `void barrier(void)` die gebruikt kan worden voor het synchroniseren van 10 threads. De eerste 9 threads die `barrier()` aanroepen dienen te blokkeren totdat een tiende thread een aanroep van `barrier()` doet. Na deze tiende aanroep komen alle threads vrij uit de barrier.

Opgave 4: MPI

(a) Wat wordt bedoeld met de SPMD (*Single Program, Multiple Data*) progammeertechniek?

(b) De MPI-routines `MPI_Send` en `MPI_Recv` hebben de volgende protoypes

```
int MPI_Send(void *buf, int count, MPI_Datatype datatype,
             int dest, int tag, MPI_Comm comm);
int MPI_Recv(void *buf, int count, MPI_Datatype datatype,
             int source, int tag, MPI_Comm comm, MPI_Status *status);
```

In een MPI-programma heeft iedere processor zijn rank nummer opgeslagen in de variabele `id`. Het totaal aantal processoren in de communicator is gegeven in `sz`. Je mag veronderstellen dat `sz` even is. Iedere processor heeft een buffer ter grootte van `N` integers. De bedoeling is om deze buffer d.m.v. een ring te communiceren naar een buurprocessor. Hiervoor wordt gebruik gemaakt van de volgende code

```
MPI_Send(buf,N,MPI_INT,(id+1)%sz,99,MPI_COMM_WORLD);
MPI_Recv(buf,N,MPI_INT,(sz+id-1)%sz,99,MPI_COMM_WORLD,&status);
```

Leg uit waarom deze methode wel werkt voor kleine waarden van `N` (bijvoorbeeld `N=1`) maar niet voor grootte waarden van `N` (bijvoorbeeld `N=1000000`). Hoe kan dit probleem opgelost worden?

(c) De onderstaande implementatie van een barrier is fout. Wat is het probleem?

```
 1  void barrier() {
 2    char dummy;
 3    int id, sz;
 4    MPI_Comm_size(MPI_COMM_WORLD, &sz);
 5    MPI_Comm_rank(MPI_COMM_WORLD, &id);
 6    if (id == 0) {
 7      for (i=1; i<sz; i++) {
 8        MPI_Send(dummy,1,MPI_CHAR,i,99,MPI_COMM_WORLD);
 9      }
10    } else {
11      MPI_Status status;
12      MPI_Recv(dummy,1,MPI_CHAR,0,99,MPI_COMM_WORLD,&status);
13    }
14  }
```

Verbeter de implementatie (uiteraard mag je niet `MPI_Barrier` gebruiken).

(d) Op iedere processor heeft de integer variabele `value` een unieke waarde. Schrijf een MPI-programmafragment dat het maximum bepaalt van deze waarden en op welke processor deze waarde voorkomt: laat één processor dit afdrukken. Je mag er vanuit gaan dat het aantal processoren een twee-macht is. Uiterard mag je geen gebruik maken van de operatie `MPI_Reduce` en iedere processor mag ten hoogste één send- en twee receive-operaties uitvoeren.