Strategier för att utnytja parallella system

Peter Kjellström – NSC
Upplysning 2010-04-27

Problemet

- Människor tänker normalt i seriella banor
- Det finns massor med seriell kod skriven
- Parallell-programmering svårt och okänt
- 2-24 cores i en vanlig desktop/server
- >1000 cores i en HPC-resurs
- Trenden är stigande...

Komplikationer

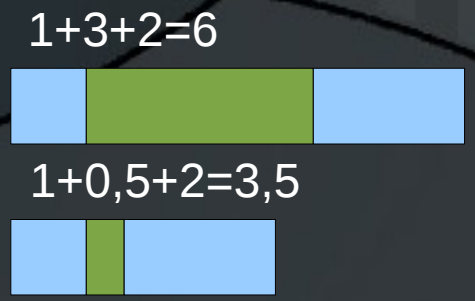
- “Strong scaling” är svårt
- Amdahls lag
- Komplicerad utvecklingsmiljö (tex. debugging/felsökning)

Strong scaling

- Kommunikationstid vs. beräkningstid
  - Det tar ~1us mellan två noder i ett kluster
  - På 1us hinner en modern core ~1000 Fops
  - Små problem hinner inte delas upp
- Strong scaling, att få ökad prestanda med fler cores med bibehållen problemstorlek
- Weak scaling, att få ökad prestanda med fler cores fast då man samtidigt ökar problemstorleken

Amdahls lag

- Det som inte (kan) parallelliseras begränsar till slut skalbarheten
- För små f har optimeringar liten effekt
- Program med många delar kräver mycket jobb



$$Speedup_{total}(f,S)=\frac{1}{(1-f)+\frac{f}{S}}$$

$$f=\frac{3}{6}=0{,}5 \quad S=6 \quad \rightarrow \frac{1}{(1-0{,}5)+\frac{0{,}5}{6}}=\frac{12}{7}$$

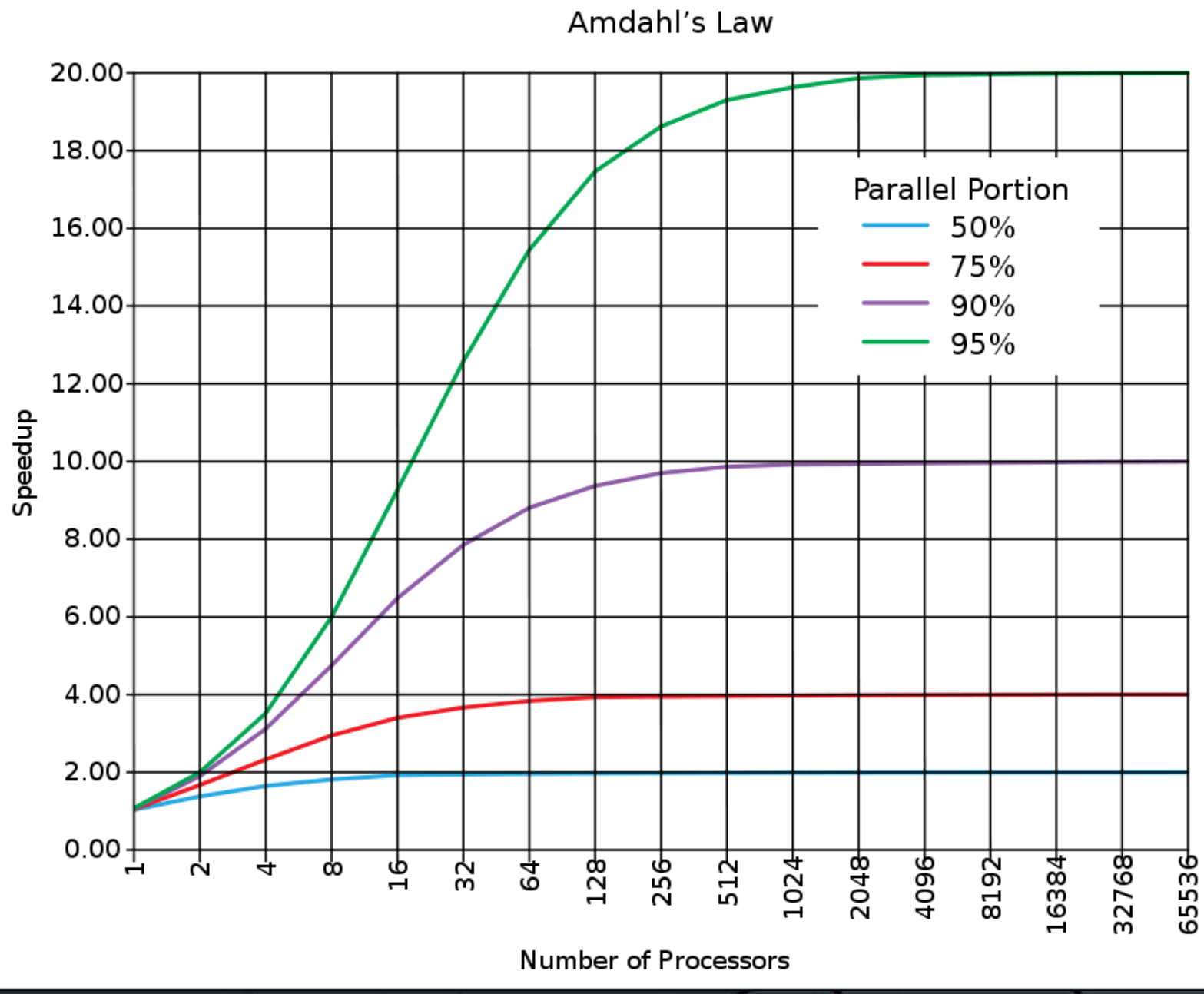


Bild från Wikipedia

Parallella strategier

- ~~Magisk mjuk-/hårdvara som fixar det~~
- Oberoende instanser, parameterstudier
- Parallella bibliotek
- Delat minne, OpenMP och Pthreads
- Distribuerat minne, MPI
- PGAS-språk

Oberoende instanser / parameterstudier / ...

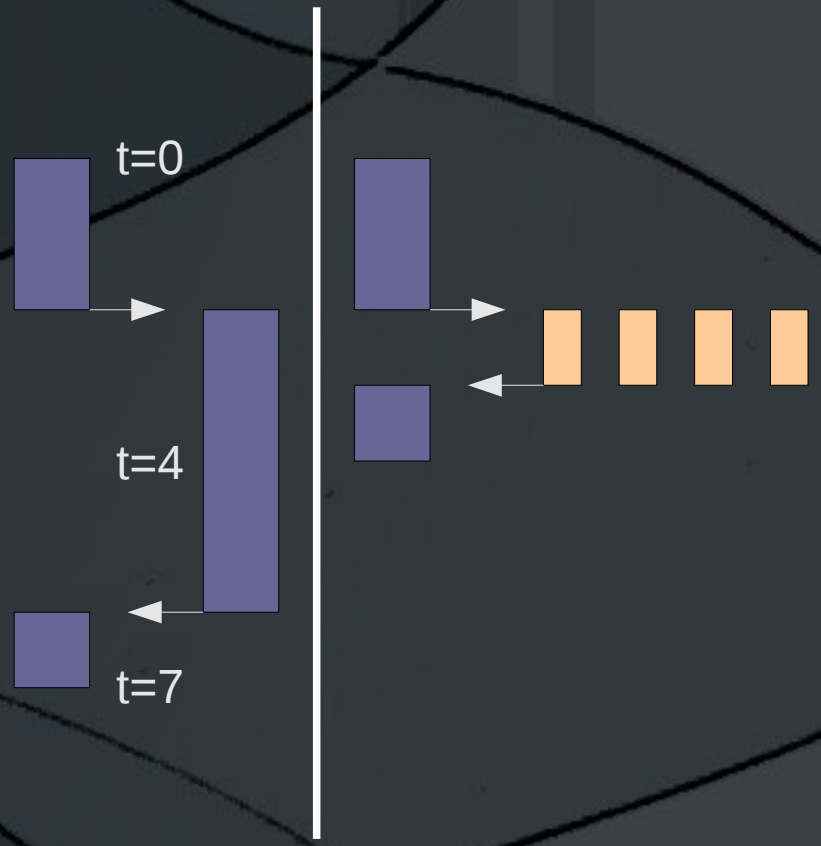
- När ditt arbete är uppdelat i oberoende delar
- Exempel
  - Web-server
  - Seti@home
  - Filmrendering
- Enkelt men funkar inte för allt/alla

```
#!/bin/sh

for core in 1 2 3 4 ; do
   ./run_my_program -variant=$core &
done

wait
```

```
#!/bin/sh

i=0
for host in x y z ; do
   for core in 1 2 3 4 ; do
      ssh $host ./render_frame --frame=$i &
      i=$(( $i + 1 ))
   done
done

wait
```

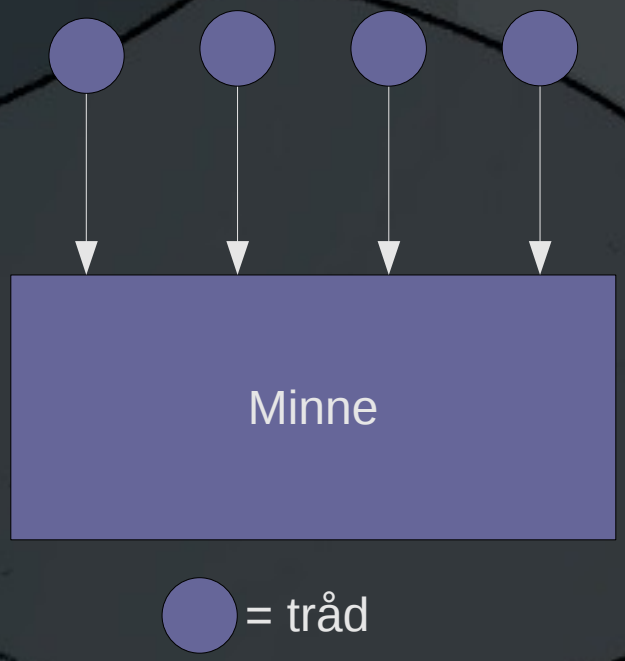
Parllella bibliotek

- Seriellt program anropar befintligt (parallelliserat) bibliotek
- Kan vara saftig lågt hängande frukt
- Amdahls lag...
- Hitta lämpligt bibliotek



Delat minne

- Flera trådar körs och har tillgång till samma minne
- Lätt att utbyta data
- Nya möjligheter till buggar, race, lås-missar etc.
- POSIX threads och OpenMP som exempel



P(osix)threads

- Biblioteksbaserat
- Flexibelt, är vad man gör det till
- Ofta delar man ut uppgifter på trådar som får göra oberoende saker
- Vanligt förekommande inom annat än HPC

```
#include <pthread.h>
#include <stdio.h>

void* func(...) {
   printf(”Hej världen”);
   pthread_exit();
}

void main(void) {
   pthread_t thread1, thread2;

   pthread_create(&thread1, func, …);
   pthread_create(&thread2, func, …);
}
```

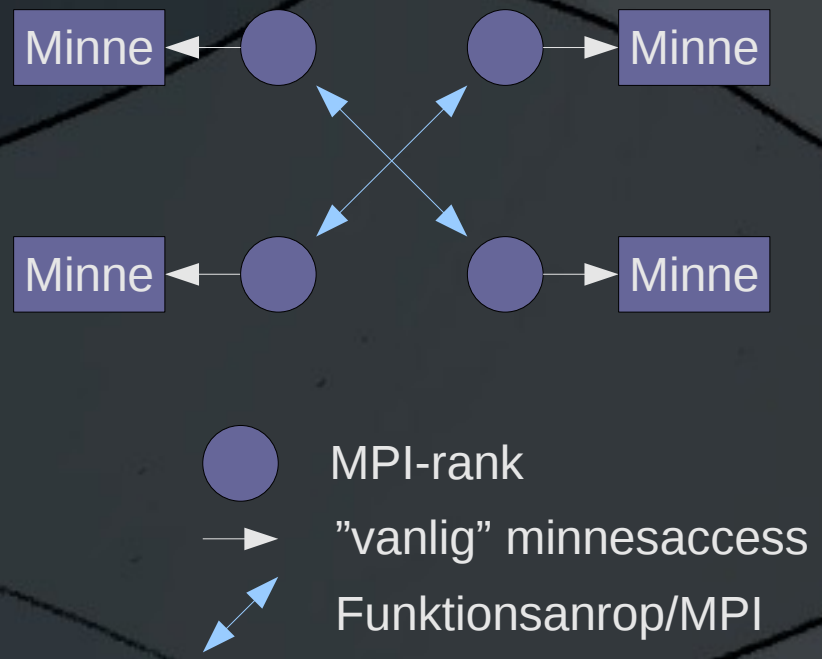
OpenMP

- Uttrycks med kompilatordirektiv
- Ofta kör trådarna varsin bit av samma loop (se exempel)
- Enkelt att göra enkla saker, svårt att göra svåra saker
- Byggs med “cc -openmp ...”

```
#include <omp.h>

int main(void) {
...
#pragma omp parallel for
  for (i=0; i<4; i++) {
    printf("thread: %i running it. %i\n",
           omp_get_thread_num(), i);
    c[i] = a[i] * b[i];
  }

  for (i=0; i<4; i++)
    printf(...

$ OMP_NUM_THREADS=4 ./test
thread: 0 running it. 0
thread: 1 running it. 1
thread: 3 running it. 3
thread: 2 running it. 2
0: a = 0, b = 10, c = 0
1: a = 1, b = 8, c = 8
2: a = 2, b = 6, c = 12
3: a = 3, b = 4, c = 12

$ OMP_NUM_THREADS=2 ./test
thread: 0 running it. 0
thread: 0 running it. 1
thread: 1 running it. 2
thread: 1 running it. 3
...
```

MPI – Message passing

- Biblioteksbaserat
- Alla trådar kör samma program
- Data skickas explicit mha. funktionsanrop
- MPI-processer kallas för rank(s) och kan köras på olika maskiner



Mer MPI...

- Omfattande utvecklingsinsats
  - Explicit uppdelning av data innebär ofta att man får skriva om stora delar av ett seriellt program
- Vanligast är att man använder Fortran eller C/C++ men tex. Python-stöd finns också
- MPI är den dominerande programmerings-modellen för HPC system idag
- MPI-program kan skala till 100 000-tals cores

MPI-exempel

```
#include <stdio.h>
#include <mpi.h>

int main(int argc, char **argv){
  int a[4], b[4], c[4];
  int i, rank, local_a, local_b, local_c;

  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);

  if (rank == 0)
    for (i=0; i<4; i++) {
      a[i] = i;
      b[i] = 10 - 2*i;
    }

  MPI_Scatter(&a, 1, MPI_INT,
              &local_a, 1, MPI_INT, 0, MPI_COMM_WORLD);
  MPI_Scatter(&b, 1, MPI_INT,
              &local_b, 1, MPI_INT, 0, MPI_COMM_WORLD);

  printf("rank: %i calculating element %i\n",
         rank, rank);
  local_c = local_a * local_b;

  MPI_Gather(&local_c, 1, MPI_INT,
             &c, 1, MPI_INT, 0, MPI_COMM_WORLD);

  if (rank == 0)
    for (i=0; i<4; i++)
      printf("%i: a = %i, b = %i, c = %i\n",
             i, a[i], b[i], c[i]);

  MPI_Finalize();
  return 0;
}
```

På NSC:

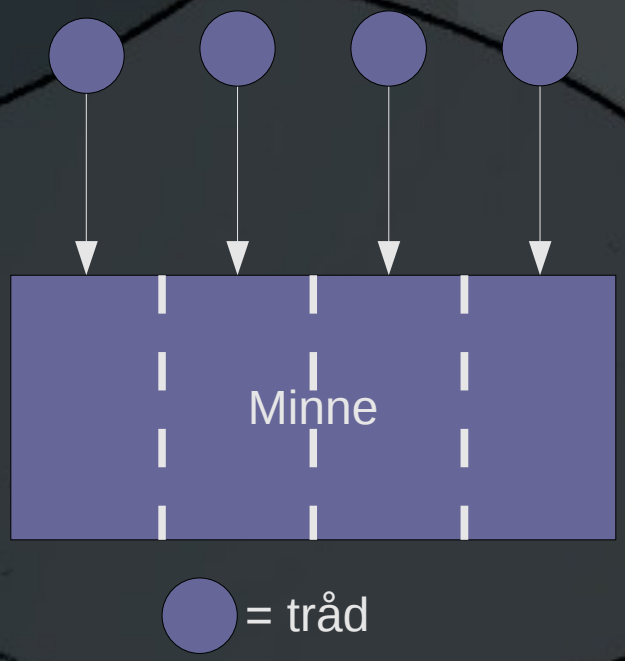
```
$ $CC -Nmpi -o mpitest mpitest.c
```

Med typisk kompilator-wrapper:

```
$ mpicc -o mpitest mpitest.c

$ mpirun -np 4 ./mpitest
rank: 0 calculating element 0
rank: 2 calculating element 2
rank: 1 calculating element 1
rank: 3 calculating element 3
0: a = 0, b = 10, c = 0
1: a = 1, b = 8, c = 8
2: a = 2, b = 6, c = 12
3: a = 3, b = 4, c = 12
```

PGAS - Partitioned Global Adress Space

- Lite som en blandning mellan MPI och OpenMP
- Implementerat som språk (eller språk-utökning)
- Exempel: X10, Unified Parallel C (UPC), CAF, Chapel, Titanium



Mer PGAS

- Lockar med mer finkornig parallellism, skalbarhet och mindre stök-o-bök än MPI
- Passar speciellt bra på maskiner med global minnesrymd (GAS) men kräver inte cache koherens (som tex. OpenMP)
- Går att köra på >1 vanlig maskin mha. GASNet
- Framtiden? Kommer folk verkligen överge MPI/OpenMP?

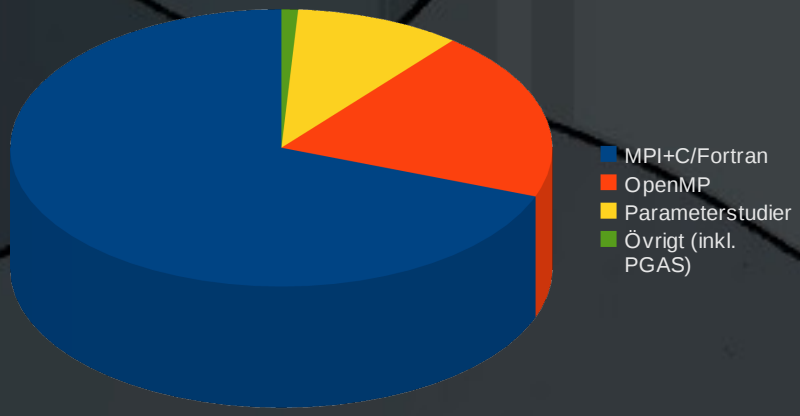
PGAS-exempel (UPC)

```
#include <upc.h>
#include <stdio.h>

shared int a[THREADS];
shared int b[THREADS];
shared int c[THREADS];

int main() {
  int i;

  if (MYTHREAD == 0)
    for (i=0; i<THREADS; i++) {
      a[i] = i;
      b[i] = 10 - 2*i;
    }

  upc_barrier;

  printf("thread: %i calculating element %i\n",
         MYTHREAD, MYTHREAD);
  c[MYTHREAD] = a[MYTHREAD] * b[MYTHREAD];

  upc_barrier;

  if (MYTHREAD == 0)
    for (i=0; i<THREADS; i++)
      printf("%i: a = %i, b = %i, c = %i\n",
             i, a[i], b[i], c[i]);

  return 0;
}
```

```
$ upcc -o test.bin test.upc

$ upcrun -n 4 test.bin
UPCR: UPC thread 2 of 4 on n139 (...
UPCR: UPC thread 3 of 4 on n139 (...
UPCR: UPC thread 1 of 4 on n139 (...
UPCR: UPC thread 0 of 4 on n139 (...
thread: 1 calculating element 1
thread: 3 calculating element 3
thread: 0 calculating element 0
thread: 2 calculating element 2
0: a = 0, b = 10, c = 0
1: a = 1, b = 8, c = 8
2: a = 2, b = 6, c = 12
3: a = 3, b = 4, c = 12
$
```

Sammanfattning

- Idag är det MPI som gäller för HPC
- Vissa metoder kan ses som lågt hängande frukt
- För ordentlig skalbarhet krävs en hel del jobb



Frågor?