Paralelné programovanie OpenMP

doc. Ing. Michal Čerňanský, PhD.
FIIT STU Bratislava

OpenMP

- OpenMP - Open Multi-Processing
- Paralelný programátorký model – explicitný paralelizmus
- Tvorba viacvláknových aplikácií v systémoch so zdieľanou pamäťou
- Poskytuje tri typy konštrukcií
  - Direktívy pre kompilátor
  - Knižničné funkcie
  - Premenné prostredia

OpenMP

- Prenositeľnosť
  - Pre C/C++ a Fortran
  - Väčšina moderných platforiem, Unix/Linux, Windows
- Štandard
  - Udržiavananý skupinou významných SW a HW dodávateľov
  - Možno ANSI štandard neskôr

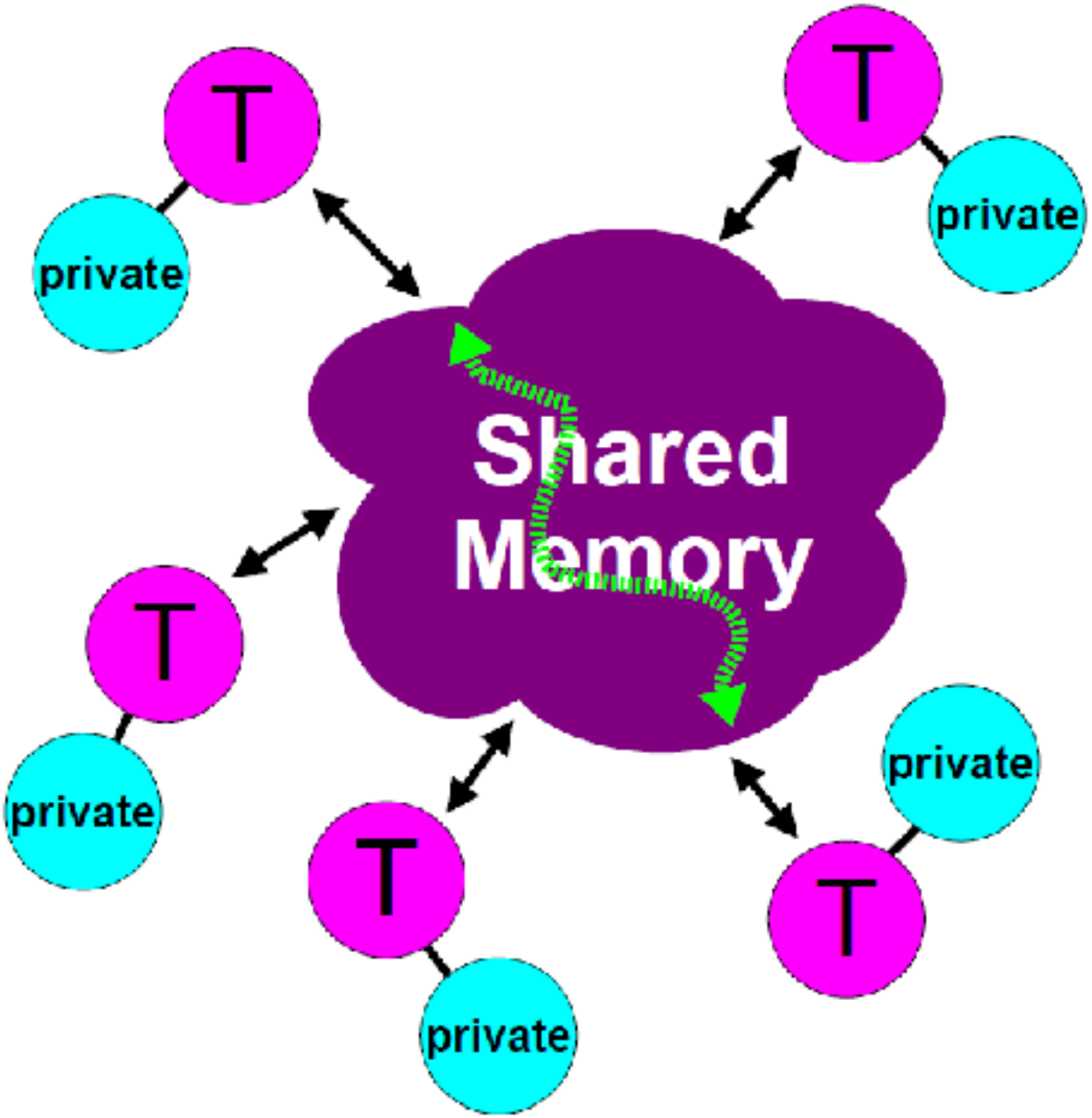
OpenMP

- Štíhle a priamo na problém zamerané rozhranie
  - Malá množina konštrukcií
  - Značný paralelizmus je môžné využiť použiim malého počtu konštrukcií
- Jednoduchosť
  - Inkrementálna paralelizácia
  - Možnosť implementovať aj hrubo aj jemnozrnný paralelizmus

OpenMP

- Ideálne riešenie pre viacjadrové počítače
- OpenMP paralelný programátorský model
  - prirodzené mapovanie pamäťového modelu a modelu vlákien na OpenMP konštrukcie
  - Odľahčený
  - Overený a zrelý
  - Akceptovaný a často používaný

OpenMP


T
private
T
private
Shared
Memory
T
private
private
T
T
private

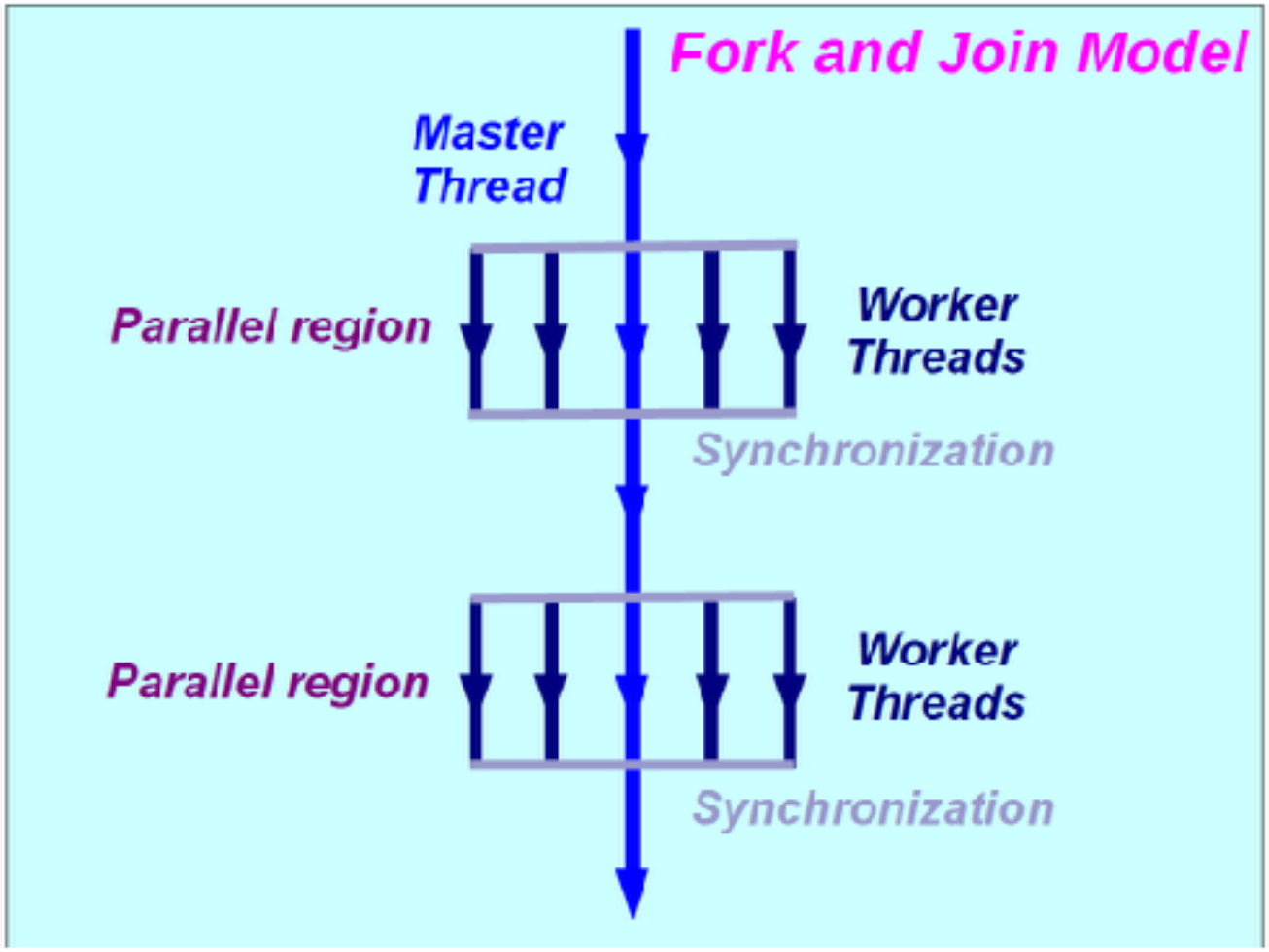
OpenMP

- Všetky vlákna majú prístup do globálnej zdieľanej pamäte
- Údaje môžu byť zdieľané medzi vláknami alebo súkromné
- Zdieľané údaje prístupné všetkými vláknami
- Súkromné premenné prístupné iba z jedného vlákna, ktoré ich vlastní
- Prenos údajov transparentný pre programátora
- Synchronizácia je väčšinou implicitná

OpenMP – zdieľanie údajov

- Údaje (premenné) je potrebne „označiť“
- Dva základné typy zdieľania:
  - Zdieľané (shared)
    - Iba jedna inštancia
    - Vlákna môžu čítať a zapisovať súčasne,
    - Iba ak chránené špeciálnymi OpenMP konštrukciami
    - Vykonané zmeny sú viditeľné pre všetky ostatné vlákna,
    - Ale nie nevyhnutne okamžite, konštrukcia „flush“
  - Súkromné
    - Každé vlákno má vlastnú kópiu údajov
    - Iné vlákno nemôže pristupovať k týmto údajom
    - Zmeny sú viditeľné iba z vlákna vlastniaceho údaje

OpenMP – model vykonávania

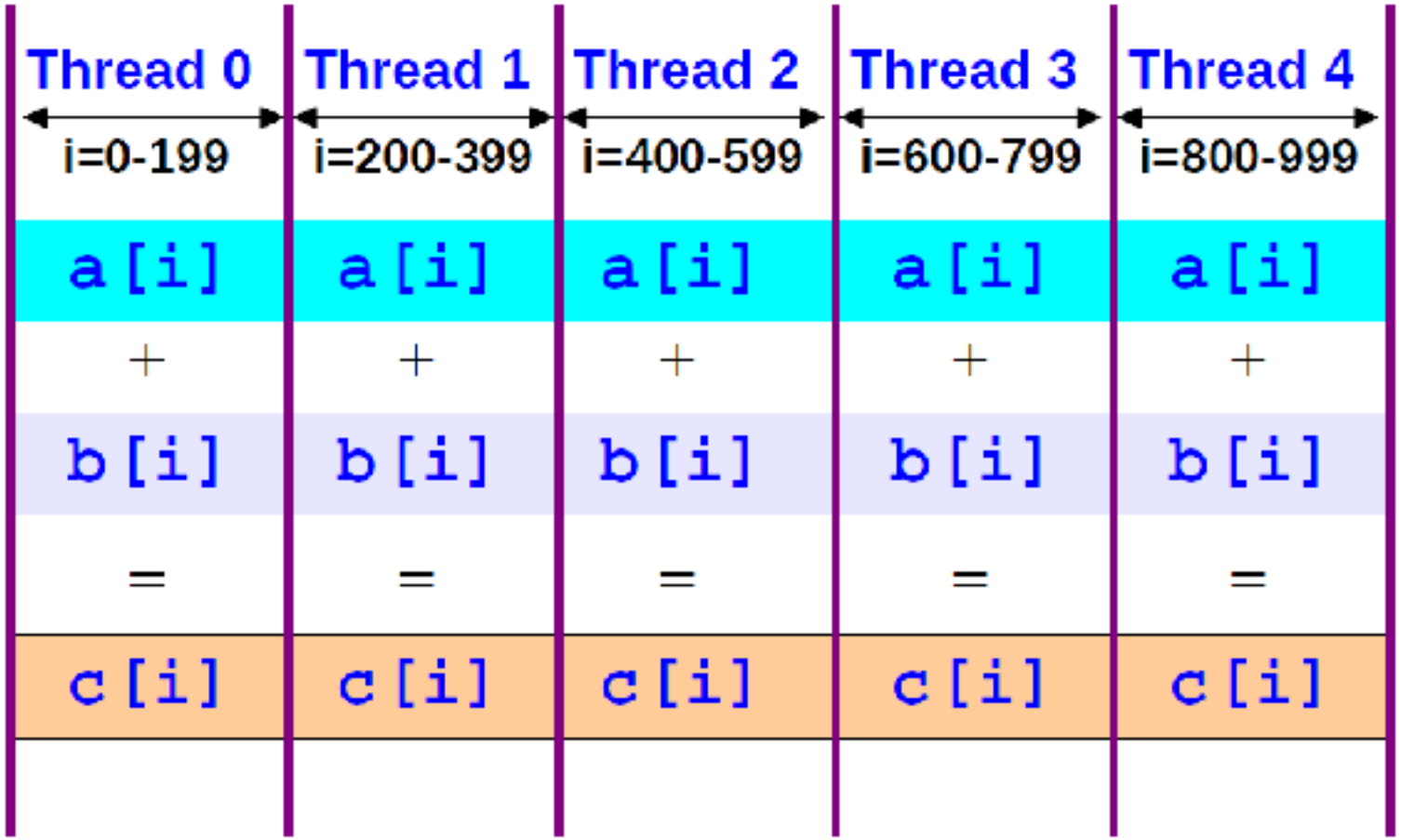


OpenMP – príklad použitia

- Paralelizácia for cyklu s použitím OpenMP direktív

```
#pragma omp parallel for
for (int i=0; i<n; i++)
    c[i] = a[i] + b[i];

#gcc myprog.c -lgomp
#set OMP_NUM_THREADS=10
#.\a.out
```

OpenMP – príklad použitia



OpenMP - konštrukcie

- Direktívy
  - Paralelná oblasť (Parallel region)
  - Rozdeľovanie práce (Worksharing)
  - Sychronizácia (Synchronization)
  - Zdieľanie údajov (Data-sharing)
    - Privátne (private)
    - firstprivate
    - lastprivate
    - zdieľané (shared)
    - Operácie redukcie (reduction)
  - Koncept siroty (Orphaning)

OpenMP - konštrukcie

- Knižničné funkcie
  - Počet vlákien
  - ID vlákna
  - Dynamická úprava počtu vlákien
  - Vnorený paralelizmus
  - Časovač
  - Uzamykanie

OpenMP - konštrukcie

- Premenné prostredia
  - Počet vlákien
  - Typ plánovania
  - Dynamická úprava počtu vlákien
  - Vnorený paralelizmus

OpenMP – konštrukcie vo vezií 3.0

- Direktívy
  - Určovanie úloh (Tasking)
- Knižničné funkcie
  - Typ plánovania
  - Aktívne úrovne (Active levels)
  - Max. počet vlákien
  - Max. úroveň vnorovania (Nesting level)
  - Rodičovské vlákno
  - Veľkosť tímu

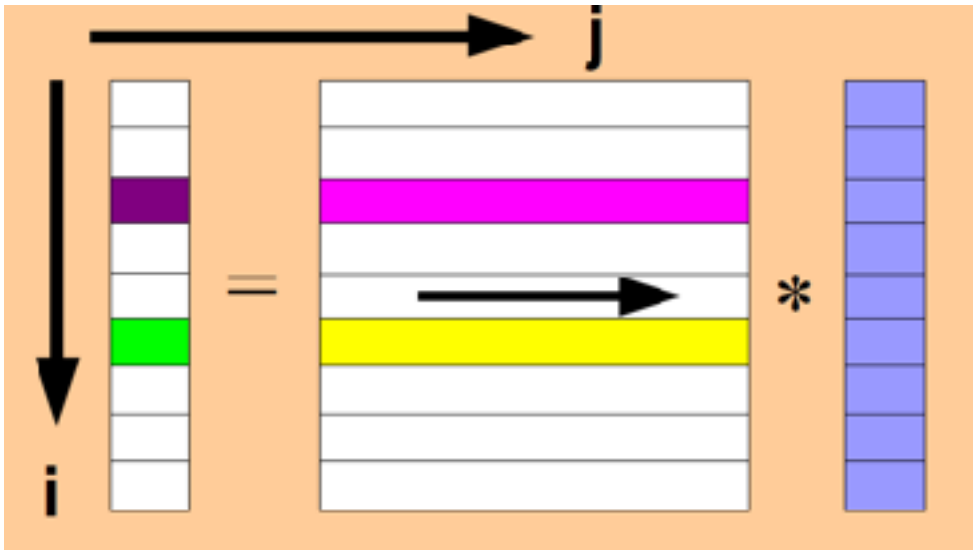
OpenMP – konštrukcie vo vezií 3.0

- Premenné prostredia
  - Veľkosť zásobníka
  - Nečinné vlákna (Idle threads)
  - Typ plánovania
  - Max. počet vlákien

OpenMP – príklad použitia

- Násobenie matice s vektorom

```
#pragma omp parallel for default(none) \
   private(i,j,sum) shared(m,n,a,b,c)
for (i=0; i<m; i++)
{
   sum = 0.0;
   for (j=0; j<n; j++)
       sum += b[i][j]*c[j];
   a[i] = sum;
}
```



OpenMP – príklad použitia

```
#pragma omp parallel if (n>limit) default(none) \
   shared(n,a,b,c,x,y,z) private(f,i,scale)
{
   f = 1.0;

   #pragma omp for nowait
   for (i=0; i<n; i++)
       z[i] = x[i] + y[i];

   #pragma omp for nowait
   for (i=0; i<n; i++)
       a[i] = b[i] + c[i];

   #pragma omp barrier

   scale = sum(a,0,n) + sum(z,0,n) + f;
}
```

Terminológia

- OpenMP Tím - Master + Workers
- Parallelná oblasť (Parallel Region)
  - všetky vlákna vykonávajú súbežne
  - Master ma vlákno s ID 0
  - Počet vlákien je upravený (ak je to povolené) iba pred vstupom
  - Klauzula "if" môže strážiť oblasť, ak sa podmienka vyhodnotí ako "false„ – sekvenčné vykonanie
- Konštrukcie deľbu práce
  - Práca v oblasti rozdelená medzi vlákna tímu

Direktívy a klauzule

- OpenMP direktívy podporujú klauzule (clauses) – upresňujú direktívu
- private(a) je klauzula pre for direktívu

  #pragma omp for private(a)
- Možnosť použitila klauzule závisí na direktíve
- V Cčku: sú direktívy „case sensitive“
- Syntax: #pragma omp directive [clause [clause] ...]
- Pokračovanie v riadku – použiť \ v pragma
- Podmienená kompilácia - _OPENMP makro

Direktívy a klauzule

- Rozsah platnosti (scope)
- Statický (Lexikálny) rozsah:
  - Zdrojový kód textovo medzi začiatkom a koncom štruktúrovaného bloku za direktívou
  - Nepresahuje viaceré funkcie či zdrojové súbory
- Osirotené direktíva (Orphaned Directive)
  - Direktíva mimo zapúzdrujúcej direktívy, definovaná mimo statického rozsahu inej direktívy
  - Presahuje viaceré funkcie aj zdrojové súbory
  - Dynamický rozsah
  - Statický rozsah + osirotené direktívy

Direktíva „Parallel“

- `#pragma omp parallel` *`[clause ...]`*

  `if` *`(scalar_expression)`*

  `private` *`(list)`*

  `shared` *`(list)`*

  `default (shared | none)`

  `firstprivate` *`(list)`*

  `reduction` *`(operator: list)`*

  `copyin` *`(list)`*

  `num_threads` *`(integer-expression)`*

  `structured_block`

Direktíva „Parallel“

- Keď vlákno dosiahne direktívu „parallel“
  - Vytvorí sa tím vlákien
  - Hlavné (master) vlákno je člen tímu a má ID 0
  - Tok vykonáva je duplikovaný od tohto okamihu a všetky vlákna ho vykonávajú
- Na konci platnosti direktívy je bariéra a od toho okamihu iba hlavné vlákno pokračuje v činnosti
- Ak vlákno skončí v paralelnej oblasti, skončia všetky vlákna

Direktíva „Parallel“

- Koľko vlákien?
- Vyhodnotenie **IF** klauzuly
- Podľa **NUM_THREADS** klauzuly
- Podľa funkcie **omp_set_num_threads()**
- Podľa **OMP_NUM_THREADS** premennej prostredia
- Zvyčajne podľa počtu jadier na uzle

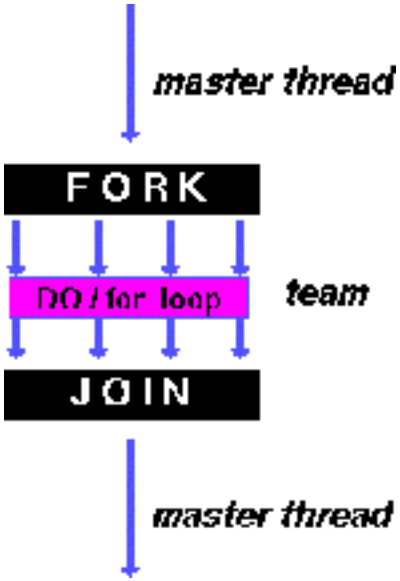
Direktíva „Parallel“

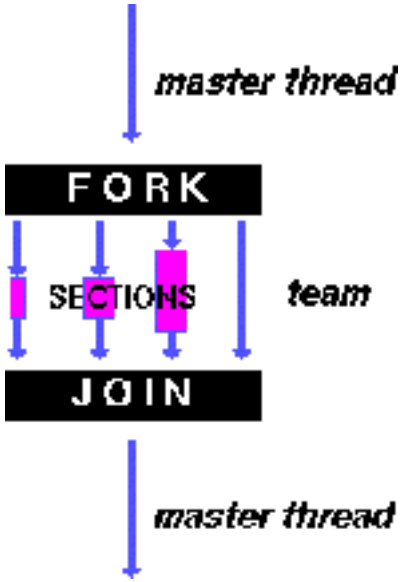
```
#include <omp.h>

main ()  {

int nthreads, tid;

/* Fork a team of threads with each thread having a private tid variable */
#pragma omp parallel private(tid)
 {

  /* Obtain and print thread id */
  tid = omp_get_thread_num();
  printf("Hello World from thread = %d\n", tid);

  /* Only master thread does this */
  if (tid == 0)
    {
    nthreads = omp_get_num_threads();
    printf("Number of threads = %d\n", nthreads);
    }
  }  /* All threads join master thread and terminate */
}
```

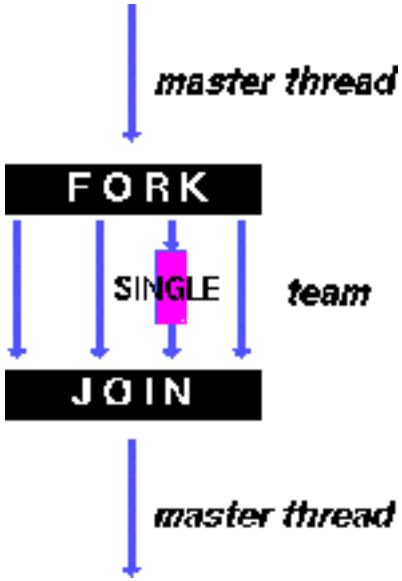
Ďeľba práce

- **Ďeľba práce (Work-Sharing Constructs)**
- Rozdelenie vykonávania oblasti medzi vlákna
- Nevznikajú nové vlákna
- Nie je automaticky bariéra pri vstupe do oblasti, ale pri výstupe áno

Ďeľba práce


master thread
FORK
DO / for loop
team
JOIN
master thread


master thread
FORK
SECTIONS
team
JOIN
master thread


master thread
FORK
SINGLE
team
JOIN
master thread

Direktíva „for“

```
#pragma omp for [clause ...]
                schedule (type [,chunk])
                ordered
                private (list)
                firstprivate (list)
                lastprivate (list)
                shared (list)
                reduction (operator: list)
                collapse (n)
                nowait

   for_loop
```

Direktíva „for“

- **Klauzula SCHEDULE**
  - **Ako sú iterácie rozdelené medzi vlákna**
  - STATIC – určené rozsahy podľa parametra *chunk* a tie statický priradené vláknam
  - DYNAMIC – iterácie rozdelené do rozsahov veľkosti *chunk* a potom dynamicky priraďované vláknam
  - GUIDED – veľkosť rozsahu daná počtom nepriradených iterácií vydelených počtom vlákien, až po veľkosť *chunk*
  - RUNTIME – podĺa premennej prostredia OMP_SCHEDULE
  - AUTO – ponechané na kompilátor

Direktíva „for“

- NO WAIT – nie je synchronizácia na konci oblasti
- ORDERED – iterácie vykonávané ako v sériovom programe
- COLLAPSE – počet vnorených cyklov použitých na vytvorenie jedného „priestoru“ iterácií

Direktíva „for“

```
#include <omp.h>
#define CHUNKSIZE 100
#define N     1000

main ()
{

int i, chunk;
float a[N], b[N], c[N];

/* Some initializations */
for (i=0; i < N; i++)
  a[i] = b[i] = i * 1.0;
chunk = CHUNKSIZE;

#pragma omp parallel shared(a,b,c,chunk) private(i)
  {

  #pragma omp for schedule(dynamic,chunk) nowait
  for (i=0; i < N; i++)
    c[i] = a[i] + b[i];
  }  /* end of parallel section */
}
```

Direktíva „sections“

```
#pragma omp sections [clause ...]
                     private (list)
                     firstprivate (list)
                     lastprivate (list)
                     reduction (operator: list)
                     nowait
  {

  #pragma omp section

     structured_block

  #pragma omp section

     structured_block

  }
```

Direktíva „sections“

```
#include <omp.h>
#define N     1000

main ()
{

int i;
float a[N], b[N], c[N], d[N];

/* Some initializations */
for (i=0; i < N; i++) {
  a[i] = i * 1.5;
  b[i] = i + 22.35;
  }
```

```
#pragma omp parallel shared(a,b,c,d)
   private(i)
  {

  #pragma omp sections nowait
    {

    #pragma omp section
    for (i=0; i < N; i++)
      c[i] = a[i] + b[i];

    #pragma omp section
    for (i=0; i < N; i++)
      d[i] = a[i] * b[i];

    }  /* end of sections */

  }  /* end of parallel section */

}
```

Direktíva „single“

```
#pragma omp single [clause ...]
                   private (list)
                   firstprivate (list)
                   nowait

     structured_block
```

Kombinované direktívy

```
#include <omp.h>
#define N       1000
#define CHUNKSIZE   100

main ()  {

int i, chunk;
float a[N], b[N], c[N];

/* Some initializations */
for (i=0; i < N; i++)
  a[i] = b[i] = i * 1.0;
chunk = CHUNKSIZE;

#pragma omp parallel for \
   shared(a,b,c,chunk) private(i) \
   schedule(static,chunk)
  for (i=0; i < n; i++)
    c[i] = a[i] + b[i];
}
```

Rozsah platnosti premenných

- PRIVATE
- FIRSTPRIVATE
- LASTPRIVATE
- SHARED
- DEFAULT
- REDUCTION
- COPYIN

Platnosť rozsahu premenných

- Klauzula private(zoznam symbolov)
  - Žiadna asociácia s pôvodnou premennou
  - Všetky referencie sú na lokálnu premennú
  - Hodnoty sú nedefinované pri vstupe a výstupe z oblasti
- Klauzula shared(zoznam symbolov)
  - Premenná prístupná všetkými vláknami tímu
  - Všetky vlákna pristupujú k rovnakej adrese

Rozsah platnosti premenných

- Klauzula firstprivate(zoznam symbolov)
  - Ako private ale hodnota inicializovaná inicializovaná z premennej platnej pred vstupom do paralelnej sekcie
- Klauzula lastprivate(zoznam symbolov)
  - Ako private ale vlákno vykonávajúce sekvenčne poslednú iteráciu alebo sekciu zapíše hodnotu lokálnej premennej do premennej platnej pred vstupom do paralelnej sekcie

Rozsah platnosti premenných

```
main()
{
  A = 10;
  #pragma omp parallel
  {
    #pragma omp for private(i) firstprivate(A) lastprivate(B)
    for (i=0; i<n; i++)
    {
      ...
      B = A + i;
      ...
    }
    C = B;
  }
} /*-- End of OpenMP parallel region --*/
```

Rozsah platnosti premenných

- Klauzula default(none|shared)
  - None
    - Žiadna implicitná predvolená hodnota rozsahu platnosti
    - Potrebné definovať rozsah všetkých premenných explicitne
  - Shared
    - Všetky premenné sú zdieľané
    - Predvolené ak nie je klauzula „default“
  - Private
    - Všetky premenné sú lokálne

Rozsah platnosti premenných

- Klauzula threadprivate(zoznam symbolov)
  - OpenMP 3.0
  - Umožňuje C++ statickým premenným byť lokálnymi premennými

```
class T {
public:
  static int i;
  #pragma omp threadprivate(i)
  ...
};
```

Rozsah platnosti premenných

- Klauzula threadprivate(zoznam symbolov)
  - Musí byť stále rovnaký počet vlákien (dynamické vlákna nepovolené)
  - Iniciálna hodnota nedefinovaná, iba ak „copyin“ klauzula použitá
- Klauzula copyin(zoznam symbolov)
  - Na začiatku paralelnej oblasti sú údaje z hlavného (master) vlákna kopírované do ostatných vlákien

Rozsah platnosti premenných

- Klauzula reduction(operator, zoznam symbolov)
  - Musí byť zdieľaná (shared) premenná
  - Operátor – asociatívna operácia
  - Iba v paralelnej oblasti a v oblastiach deľby práce
  - Premenná iba v definovaných kontextoch:

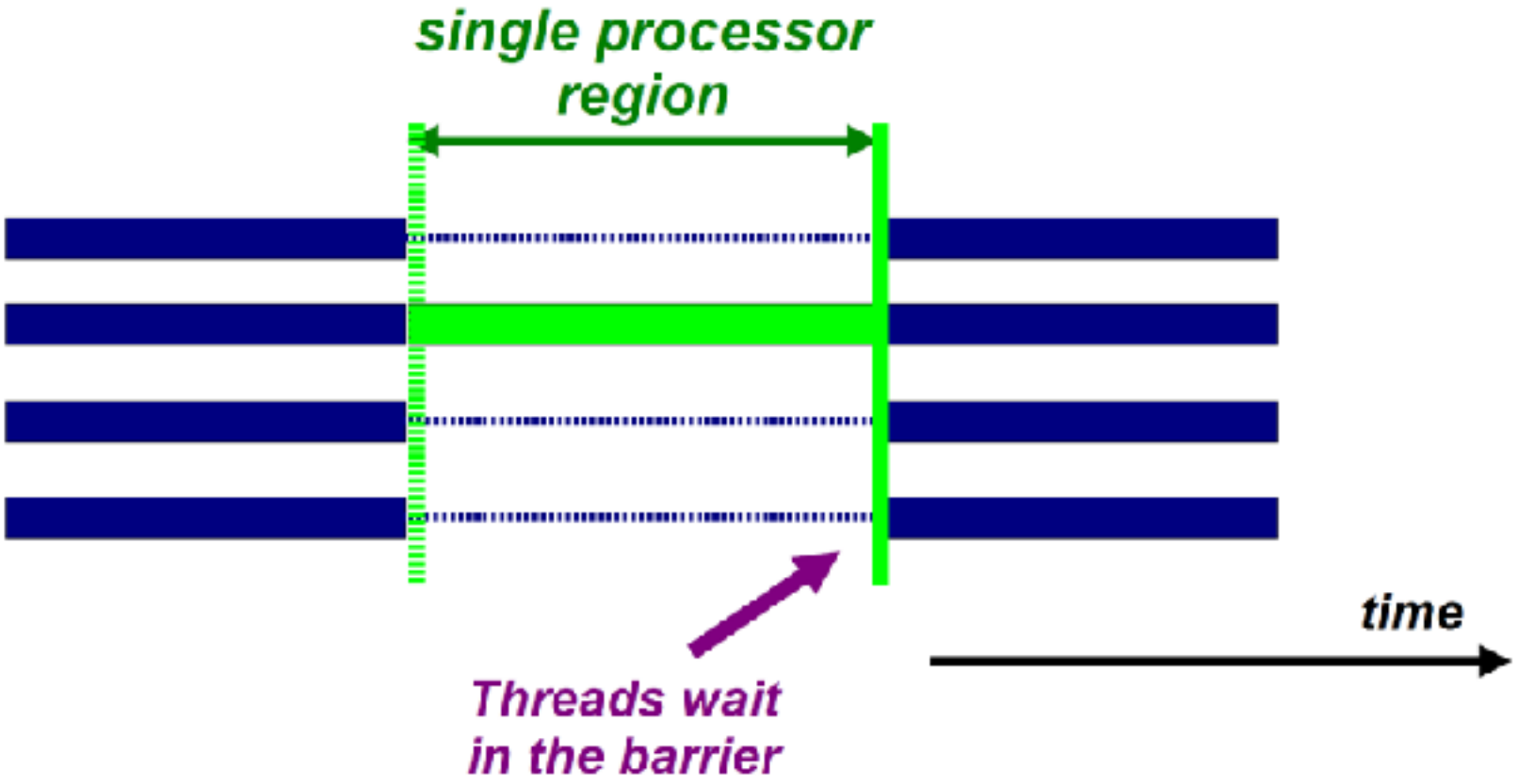
```
x = x op expr
x = expr op x (except subtraction)
x binop = expr
x++
++x
x--
--x
```

Klauzula „reduction“

```
#include <omp.h>

main ()  {

int   i, n, chunk;
float a[100], b[100], result;

/* Some initializations */
n = 100;
chunk = 10;
result = 0.0;
for (i=0; i < n; i++)
  {
  a[i] = i * 1.0;
  b[i] = i * 2.0;
  }
```

```
#pragma omp parallel for       \
  default(shared) private(i)   \
  schedule(static,chunk)       \
  reduction(+:result)

  for (i=0; i < n; i++)
    result = result + (a[i] *
   b[i]);

printf("Final result=
   %f\n",result);

}
```

Direktíva „master“ a „single“



Direktíva „critical“

- Oblasť vykonávaná iba jedným vláknom v danom čase
- #pragma omp critical [name]
- Rôzne oblasti s rovnakým menom – ako jedna oblasť
- Bez mena – rovnaká oblasť

Direktíva „flush“

- #pragma omp flush (zoznam premenných)
- Synchronizačný bod, po ktorom implementácia poskytne konzistentný pohľad na premenné
- Aj v „cahche - coherent“ systémoch

Direktíva „barrier“

- #pragma omp barrier

- Všetky vlákna v tíme sa zosynchronizujú
- Vlákno čaká, pokiaľ všetky vlákna nedosiahnu bariéru

Osirotenie (Orphaning)

- OpenMP špecifikácia nevyžaduje, aby direktívy deľby práce alebo synchronizačné direktívy (omp for, single, critical, barrier) boli umiestnené v rámci lexikálneho rozsahu paralelnej oblasti (omp parallel)
- Ak sú mimo paralelnej oblasti, sú „osirotené“ a v sekvenčnej časti programu vykonávané iba hlavným vláknom (direktíva je ignorovaná)

Knižničné funkcie

- Počet vlákien
- ID vlákna
- Dynamická úprava počtu vlákien
- Vnorený paralelizmus
- Časovač
- Uzamykanie

Knižničné funkcie – počet vlákien

- #include <omp.h>
- void omp_set_num_threads(int num_threads)

- Ak dynamické určovanie poćtu vlákien – max. počet vlákien pre paralelnú oblasť, inak presný počet vlákien pre paralelnú oblasť, do najbližšieho volania tejto funkcie
- Funkcia môže byť volaná iba zo sekvenčnej časti programu

Knižničné funkcie – počet vlákien

- #include <omp.h>
- int omp_get_num_threads(void)

- Počet vlákien v tíme práve vykonávanej paralelnej oblasti, v kt. je f. volaná

Knižničné funkcie – počet vlákien

- #include <omp.h>
- int omp_get_max_threads(void)

- Maximálna hodnota, kt. môže get_num_threads vrátiť

Knižničné funkcie – počet vlákien

- #include <omp.h>
- int omp_get_thread_num(void)

- Vráti číslo vlákna v rámci tímu, hlavné vlákno má číslo 0

Knižničné funkcie – počet vlákien

- #include <omp.h>
- int omp_get_thread_limit(void)

- OpenMP 3.0
- Maximálny možný počet vlákien pre program

Knižničné funkcie

- #include <omp.h>
- int omp_get_num_proc(void)

- Počet dostupných procesorov

Knižničné funkcie

- #include <omp.h>
- int omp_in_parallel(void)

- Zistenie, či je sekcia vykonávaná paralelne

Knižničné funkcie

- #include <omp.h>
- void omp_set_dynamic(int dynamic_threads)
- int omp_get_dynamic(void)

- Povolenie a zakázanie dynamickej úpravy počtu vlákien
- Predvolené správanie je implementačne závislé
- Musí byť volané zo sekvenčnej časti programu

Knižničné funkcie

- #include <omp.h>
- void omp_init_lock(omp_lock_t *lock)
- void omp_init_nest_lock(omp_nest_lock_t *lock)

- Inicializácia zámku asociovaného s premennou
- Iniciálny stav je „zamok odomknutý“

Knižničné funkcie

- #include <omp.h>
- void omp_init_lock(omp_lock_t *lock)
- void omp_init_nest_lock(omp_nest_lock_t *lock)
- void omp_destroy_lock(omp_lock_t *lock) void omp_destroy_nest__lock(omp_nest_lock_t *lock)

- Povolenie a zakázanie vnoreného paralelizmu
- Predvolené správanie – vnorený paralelizmus je zakázaný

Knižničné funkcie

- #include <omp.h>
- void omp_set_lock(omp_lock_t *lock)
- void omp_set_nest__lock(omp_nest_lock_t *lock)
- void omp_unset_lock(omp_lock_t *lock)
- omp_unset_nest__lock(omp_nest_lock_t *lock)
- int omp_test_lock(omp_lock_t *lock)
- int omp_test_nest__lock(omp_nest_lock_t *lock)
- Čakanie a uzamknutie zámku
- Uvoľnenie zámku
- Testovanie zámku

Knižničné funkcie

- #include <omp.h>
- double omp_get_wtime(void)

- Počet uplynutých sekúnd od minulého obdobia – použitie v pároch

- Prenositeľná funkcia

Knižničné funkcie

- #include <omp.h>
- double omp_get_wtick(void)

- Počet sekúnd medzi dvoma taktami procesora

Premenné prostriedia

- OMP_SCHEDULE
  - Direktíva for so schedule klauzulou nastavenou na „runtime“
  - setenv OMP_SCHEDULE "guided, 4" setenv OMP_SCHEDULE "dynamic"
- OMP_NUM_THREADS
  - Nastavenie max. počtu vlákien
  - setenv OMP_NUM_THREADS 8
- OMP_DYNAMIC
  - Povolenie/zakázanie dynamickej úpravy počtu vlákien
  - setenv OMP_DYNAMIC TRUE
- OMP_NESTED
  - Povolenie/zakázanie vnoreného paralelizmu
  - setenv OMP_NESTED TRUE

Premenné prostriedia

- OMP_STACKSIZE
  - Veľkosť zásobníka (OpenMP 3.0)
- OMP_WAIT_POLICY
  - Pasívne alebo aktívne čakanie (OpenMP 3.0)
  - setenv OMP_WAIT_POLICY active
setenv OMP_WAIT_POLICY passive
- OMP_MAX_ACTIVE_LEVELS
  - Max. počet vnorených aktívnych paralelných oblastí (OpenMP 3.0)
  - setenv OMP_MAX_ACTIVE_LEVELS 2
- OMP_THREAD_LIMIT
  - Max. počet vlákien pre program (OpenMP 3.0)
  - setenv OMP_THREAD_LIMIT 8

Práca s úlohami (Tasking)

- Novinka vo verzii 3.0
- Umožňuje paralelizaciu širšej množiny úloh

- #pragma omp task [clauses]
  - Definícia novej úlohy

- #pragma omp taskwait
  - Aktuálna úloha preruší vykonávanie, kým všetky dcérske úlohy vytvorené aktuálnou úlohou nie sú ukončené

Práca s úlohami (Tasking)

- Ťažké pre OpenMP 3.0
- Spočítať iterácie
- Spraviť z while cyklu for cyklus

```
while(my_pointer)  {

   (void) do_independent_work (my_pointer);
   my_pointer = my_pointer->next ;

} // End of while loop
```

Práca s úlohami (Tasking)

```
my_pointer = listhead;

#pragma omp parallel
{
  #pragma omp single nowait
  {
    while(my_pointer) {
      #pragma omp task firstprivate(my_pointer)
      {
        (void) do_independent_work (my_pointer);
      }
      my_pointer = my_pointer->next ;
    }
  } // End of single - no implied barrier (nowait)
} // End of parallel region - implied barrier
```

Práca s úlohami (Tasking)

```
long comp_fib_numbers(int n) {

  // Basic algorithm: f(n) = f(n-1) + f(n-2)
  long fnm1, fnm2, fn;
  if ( n == 0 || n == 1 ) return(n);

  fnm1 = comp_fib_numbers(n-1);
  fnm2 = comp_fib_numbers(n-2);

  fn = fnm1 + fnm2;

  return(fn);
}
```

Práca s úlohami (Tasking)

```
long comp_fib_numbers(int n){
  // Basic algorithm: f(n) = f(n-1) + f(n-2)
  long fnm1, fnm2, fn;
  if ( n == 0 || n == 1 ) return(n);

  #pragma omp task shared(fnm1)
  {fnm1 = comp_fib_numbers(n-1);}

  #pragma omp task shared(fnm2)
  {fnm2 = comp_fib_numbers(n-2);}

  #pragma omp taskwait
  fn = fnm1 + fnm2;
  return(fn);
}
```

Práca s úlohami (Tasking)

```
#pragma omp parallel shared(nthreads)
{
  #pragma omp single nowait
  {
    result = comp_fib_numbers(n);
  } // End of single
} // End of parallel region
```

Záver

- OpenMP – jednoduchý, ale výkonný paralelný programátorský model
- Pre systémy so zdieľanou pamäťou každej veľkosti, napr. aj jednoprocesorový viacjadrový počítačový systém
- Podporovaný mnohými kompilátormi
- Verzia 3.0 na vzostupe
- Práca s úlohami – významná vlastnosť novej verzie

Zdroje

- openmp.org
- https://computing.llnl.gov/tutorials/openMP