Lliurament LAB 2 PAP per Victor Correal Ramos

En aquest document s'explica l'implementació del itinerari 1: worksharing model.

Seguidament veiem els programes que s'han utilitzat per comprobar el funcionament i les conclusions després de fer l'implementació.

Decisions d'implementació

Primerament, la llibreria inicialitza les estructures de dades de totes les possibles parts implicades i crea el thread amb identificador -1.

La primera estructura que em de construir es la parallel; esta pensat com la unitat de control dels threads. La llibreria inicialitza el thread amb identificador -1 i totes les crides a GOMP_* depenen d'aquest thread. El controlador guarda per cada thread:

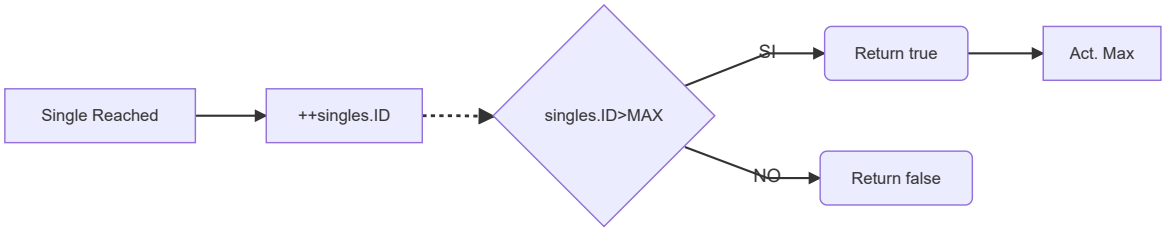
- les dades i codi que s'ha d'executar.
- l'identificador de thread d'intre del 'team' i nivell d'anidat.
- Una barrera i un punter a barrera (només un thread de cada team crearà la barrera).
- Array d'enters, per identificar els joins que s'han de fer.

Amb aquest controlador podem implementar les crides a parallel i asegurar que els paralels anidats acabaran abans que el paralel exterior. Per tal de permetre les regions anidades, les crides a funcions intrínseques es fan contra la clau especifica de cada thread.

Pel que fa a les estructures de sincronització:

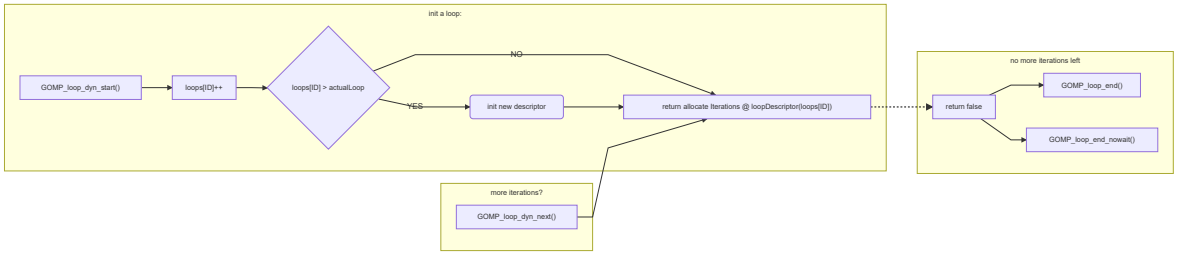
- La barrier esta construida amb una pthread barrier i s'actualitzen el numero de threads a esperar al inici de cada regió paral·lela.
- Les seccions crítiques sense nom, s'implementen directament amb un pthread_mutex
- Les seccions crítiques amb nom estan implementades sobre una estructura de tipus map, amb un màxim de 5 named locks. S'ha utilitzat una estructura d'un array per fer el map.

Seguidament, el constructor single esta implementat de forma 'arriscada'. El controlador d'aquesta estructura es guarda, per cada thread, el número de vegades que s'ha notificat l'arribada d'un single i s'actualitza, de forma atómica una variable que representa el numero màxim de 'singles' que s'han registrat. El seu funcionament segueix aquest esquema:



Es arriscat, ja que es pot retornar fals si el valor del máxim s'actualitza en més d'una unitat.

Per últim, la construcció loop guarda una llista (implementada com una llista doblement encadenada) de descriptors de loop i una estructura de control per poder saber, per cada thread, en quin loop es troba. La lógica es podria resumir amb el següent esquema:



El control coneix en cada moment i per cada thread, en quin loop es troba i pot adjudicar regions d'iteracions en cada moment.

En el moment de definir el descriptor, es calculen els chunks necesaris i un array permet saber quines regions han sigut adjudicades ja. Els procediments de dynamic_start i allocate_iterations es fa protegit amb mutex, ja que hi ha condicions de carrera sobre la llista global de descriptors.

A més, a l'implementació s'ha afegit una compilació condicionada, per poder generar missatges de traça. Cal compilar amb el simbol _DEBUG=1 o cridar a make dbg=1 per compilar la llibreria.

Possibles millores

Hi ha certes opcions de diseny, que podrien aportart millor rendiment peró que son mes complicades d'implementar:

1. Hash map per les seccions crítiques amb nom.
2. Adjudicació d'iteracions sobre "bancs" de threads, aixi l'exclusió mutua només afecta al banc d'aquell thread.
3. Cercant de noves iteracions amb un valor aleatori
4. Afegir les seccions crítiques amb nom al descriptor de la regió paral·lela

Resultats

S'ha definit una macro (LOG) per tal de poder veure el funcionament global i coordinat de cada part de la llibreria.

Els programas de prova que s'han utilitzat, i els resultats comparats amb la versió real es resumeixen en aquesta taula.

Test	Resultat	Descripció
tparallel	Correcte	Diferents regions paral·leles amb diferents threads
tbarrier	Correcte	Barreres consecutives
tbarrier1	Correcte	Barreres consecutives, en diferents regions paral·leles
tsynch	Correcte	Diferents barreres i seccions critiques
tsingle1	Correcte	Un unic single
tsingle2	Correcte	Dos single nowait amb el máxim de threads possibles
tsingle3	Correcte	Bucle(de 1000) amb single no wait, amb el 24 threads
tworkshare	Correcte	Diversos tipus de for consecutius
tloop1	Correcte	Dos bucles amb barreres i single
tloop2	Correcte	Mes complicat que tloop2
tloop3	Correcte	Diverses operacions dinter dels bucles, amb sincronització
tmandel1	Correcte	S'han comparat les dos sortides del executable i són correctes

El rendiment que veiem al tmandel es presenta a la seguent taula (temps en segons):

	Temps d'execucció	**Threads**
GOMP	1,417	5
myOMP	1,421	5
Escalabilitat(myOMP)	**Escalabilitat(GOMP)**	
1,688	1,687	2
1,418	1,412	4
1,411	1,411	8
1,424	1,414	10
1,434	1,411	12
1,431	1,436	14
1,428	1,424	16
1,427	1,412	18
1,433	1,421	20

S'han realitzat 5 execuccions i s'ha prés la mitjana.

Conclusions i valoracions

Un cop acabat la meva modesta implementació d'openMP he pogut aprende molt millor les eines que proporciona POSIX threads. També el funcionament de la llibreria real d'openMP.

Els resultats son força próxims a l'implementació real, tot i que n'hi han situacions de certes sentències pragma que fan que la meva implementació caigui en un bucle infinit d'espera o problemes similars.

Malgrat els problemes puc concloure que realitzat una implementació força funcional. A millorar potser