Kvarkový pohon

Termín odevzdání:	**06.04.2014 23:59:59**
Hodnocení:	**33.0000**
Max. hodnocení:	**30.0000** (bez bonusů)
Odevzdaná řešení:	3 / 50
Nápovědy:	0 / 0

Úkolem je realizovat sadu funkcí, které budou optimalizovat využití energie v kvarkových motorech.

Autoři seriálu StarTrek vymysleli mnoho zajímavých způsobů, jak pohánět kosmické lodi. Příkladem jsou motory využívající dilithiové krystaly, anihilace, kvantové singularity nebo červené hmoty. V předmětu OSY autorům pomůžeme a navrhneme koncept motorů využívajících fúze kvarků. Kromě tohoto revolučního nápadu navíc dodáme i software, který umožní rychlé, efektivní a ekologické řízení takových motorů.

Základních kvarků je 6 (viz **Quark**). Názvy kvarků (horní, dolní, podivný, půvabný, pravdivý a krásný) se nám pro strojové zpracování moc nehodí, názvy proto nahradíme čísly 0 až 5. Předpokládáme, že dvojice kvarků je schopna (za určitých podmínek) reagovat. Ze dvojice kvarků vznikne jeden jiný (nebo i stejný) kvark a uvolní se energie. Subatomární částice se chovají podivně, není tedy překvapivé, že fúze jedné dvojice kvarků může poskytovat různé výstupy s různou energií. Například fúze kvarků 0 + 1 může poskytnout výsledný kvark 0 s energií x, nebo např. kvark 4 s energií y. K fúzi dochází pouze v generátorech, konfigurace generátoru je přesně popsána symetrickou maticí 6 x 6 x 6 čísel (3D pole). První a druhý index udává vstupující kvarky, třetí index udává vznikající kvark. Pokud tedy probíhá fúze x + y na z, bude množství uvolněné energie udáno obsahem pole [x][y][z]. Pokud na dané pozici je kladné číslo (energie se uvolní), může v tomto generátoru taková reakce proběhnout, pokud je na pozici hodnota 0, znamená to, že reakce proběhnout nemůže (alespoň ne v tomto generátoru). Generátorů může být k dispozici více, každý generátor bude popsán svojí maticí 6 x 6 x 6 hodnot.

Palivo do motorů je tvořeno kvarky, které jsou pro snadné skladování uložené v kruhových strukturách - palivových kruzích. Palivový kruh vložen do jednoho vybraného generátoru a v něm je přeměněn na energii. Generátor pracuje tak, že vybírá dvojici sousedních kvarků, provede požadovanou fúzi (výsledek dle volby, ale omezen podle své matice) a výsledný kvark umístí na uvolněné místo. Pak vybere další dvojici sousedních kvarků, provede další fúzi a celý proces se opakuje. Zpracování palivového kruhu skončí v okamžiku, kdy zbude poslední jeden kvark (ten je jako odpad vypuštěn).

Na projektu kvarkového motoru se bohužel podílely strukturální fondy Evropské Unie. Závazná směrnice EUROQUARK V udává, že:

- palivový kruh musí být celý zpracovaný v jednom generátoru (v průběhu jej nelze přenášet mezi generátory),
- palivový kruh musí být spotřebovaný celý, tedy výstupem musí být právě jeden kvark,
- výsledný kvark musí být likvidován ekologicky s ohledem na udržitelný rozvoj, např. je zcela kategoricky zakázáno vypouštět v Bruselu podivné a pravdivé kvarky. Proto je požadavkem, aby bylo možné řídit, který výsledný kvark vznikne.

Vaším úkolem je realizovat software, který dokáže kvarkové motory řídit. Předpokládejte, že dostanete popis dostupných generátorů (pro každý jeho popisnou matici) a dále předpokládejte, že dostáváte popis zpracovávaných palivových kruhů. Pro každý kruh dostanete jeho velikost (počet kvarků), seznam kvarků v něm umístěných a požadovaný výsledný kvark. Váš program musí rozhodnout, který generátor dokáže palivový kruh zpracovat a jakou energii lze z kruhu získat. Pokud je možností více, musí vybrat ten generátor, který poskytne největší energii. Výstupem je identifikace použitého generátoru, uvolněná energie a strom popisující postupné fúze kvarků v kruhu. Výpočet chceme samozřejmě co nejrychlejší, protože se jedná o úlohu výpočetně náročnou, hodí se využít výpočetní výkon co nejvíce CPU jader. K tomu využijete vlákna.

Požadované rozhraní:

```
#define QUARKS_NR     6

struct TGenerator
 {
   unsigned int m_Energy[QUARKS_NR][QUARKS_NR][QUARKS_NR];
 };

struct CReactNode
 {
                   CReactNode       ( uint8_t           product,
                                      unsigned int      energy,
                                      CReactNode      * l = NULL,
                                      CReactNode      * r = NULL );
                  ~CReactNode       ( void );
   int             PrintTree        ( std::ostream    & os,
                                      int               pos,
                                      int               len,
                                      const std::string & nodePrefix = "",
                                      const std::string & subNodePrefix = "" );
   CReactNode    * m_L;
   CReactNode    * m_R;
   unsigned int    m_Energy;
   uint8_t         m_Product;
```

```
 };

struct TRequest
 {
   uint8_t       * m_Fuel;
   int             m_FuelNr;
   uint8_t         m_FinalProduct;
 };

struct TSetup
 {
   CReactNode    * m_Root;
   unsigned int    m_Energy;
   int             m_Generator;
   int             m_StartPos;
 };

void               optimizeEnergySeq( const TGenerator* generators,
                                      int               generatorsNr,
                                      const TRequest  * request,
                                      TSetup          * setup );

void               optimizeEnergy   ( int               threads,
                                      const TGenerator* generators,
                                      int               generatorsNr,
                                      const TRequest *(* dispatcher)( void ),
                                      void           (* engines ) ( const TRequest *, TSetup * ) );
```

`TGenerator`
: struktura popisuje konfiguraci generátoru. Jedná se o zabalené 3D pole, prvé 2 indexy udávají slučované kvarky, třetí index udává výsledný kvark. Obsahem pole je buď 0 (takovou fúzi nelze provést) nebo kladné číslo (uvolněná energie).

`CReactNode`
: třída popisující strom fúzí. Listy představují kvarky z palivového kruhu, vnitřní uzly pak produkty fúzí. Členská proměnná `m_Energy` udává celkovou energii uvolněnou při fúzi podstromu, proměnná `m_Product` kvark získaný fúzí podstromu. Tato třída je implementovaná v testovacím prostředí a v dodaném archivu, Váš program ji bude pouze používat.

`TRequest`
: Struktura popisující požadavek na fúzi palivového kruhu. Obsahuje složku `m_Fuel` (pole kvarků obsažených v palivovém kruhu), `m_FuelNr` (počet kvarků) a `m_FinalProduct` (požadovaný kvark, který má vzniknout po syntéze celého kruhu). Strukturu budete dostávat z testovacího prostředí a budete ji v nezměněné podobě (stejná adresa, stejný obsah) vracet.

`TSetup`
: je struktura popisující fúzi palivového kruhu. Má složky:

  - `m_Root` - strom fúzí, viz níže. Pokud lze fúzi s danými požadavky provést, bude `m_Root` nastaven na kořen stromu popisujícího postupné fúze sousedů. Pokud fúzi nelze realizovat, bude nastaven na `NULL`,
  - `m_Energy` - celková uvolněná energie při fúzi, 0 pokud nelze fúzi realizovat,
  - `m_Generator` - index generátoru, který má být použit,
  - `m_StartPos` - index kvarku z pole `m_Fuel`, který odpovídá nejlevějšímu listu ve stromu `m_Root`. Význam je lépe vidět na příkladu.

`optimizeEnergySeq`
: Vámi implementovaná funkce, která sekvenčně vypočte optimální fúzi zadaného palivového kruhu. Funkce dostane parametry:

  - `generators/generatorsNr` - popisy jednotlivých dostupných generátorů,
  - `request` - popis palivového kruhu, pro který má být proveden výpočet,
  - `setup` - výstupní parametr, do kterého funkce umístí výsledky výpočtu.

  Funkce je určena pro odladění algoritmu výpočtu. Dokud tato funkce nebude fungovat, testovací prostředí nebude Váš program dále testovat v testech s více vlákny.

`optimizeEnergy`
: Vámi implementovaná funkce, která vypočte optimální fúze zadaných palivových kruhů s využitím vláken. Funkce dostane parametry:

  - `threads` - udává počet vláken, která mají být použita pro paralelní výpočet optimálních řešení,
  - `generators/generatorsNr` - popisy jednotlivých dostupných generátorů,
  - `dispatcher` ukazatel na funkci, která bude dodávat problémy k řešení. Vaše implementace bude opakovaně volat tuto funkci, každé zavolání funkce dodá popis dalšího palivového kruhu ke zpracování. Pokud volání této funkce vrátí hodnotu NULL, znamená to, že palivo došlo a není co dále zpracovávat. Vaše funkce `optimizeEnergy` v takovém případě dokončí rozdělanou práci a vrátí se.
  - `engines` ukazatel na funkci, kterou budete volat po zpracování každého jednoho palivového kruhu. Funkce má dva parametry - popis požadavku jak byl předán funkcí `dispatcher` a Vámi vyplněnou strukturu s nalezeným optimálním zpracováním.

Ukázka výpočtu:

```
Matice generátoru:
          0                   1                   2                   3                   4                   5
```

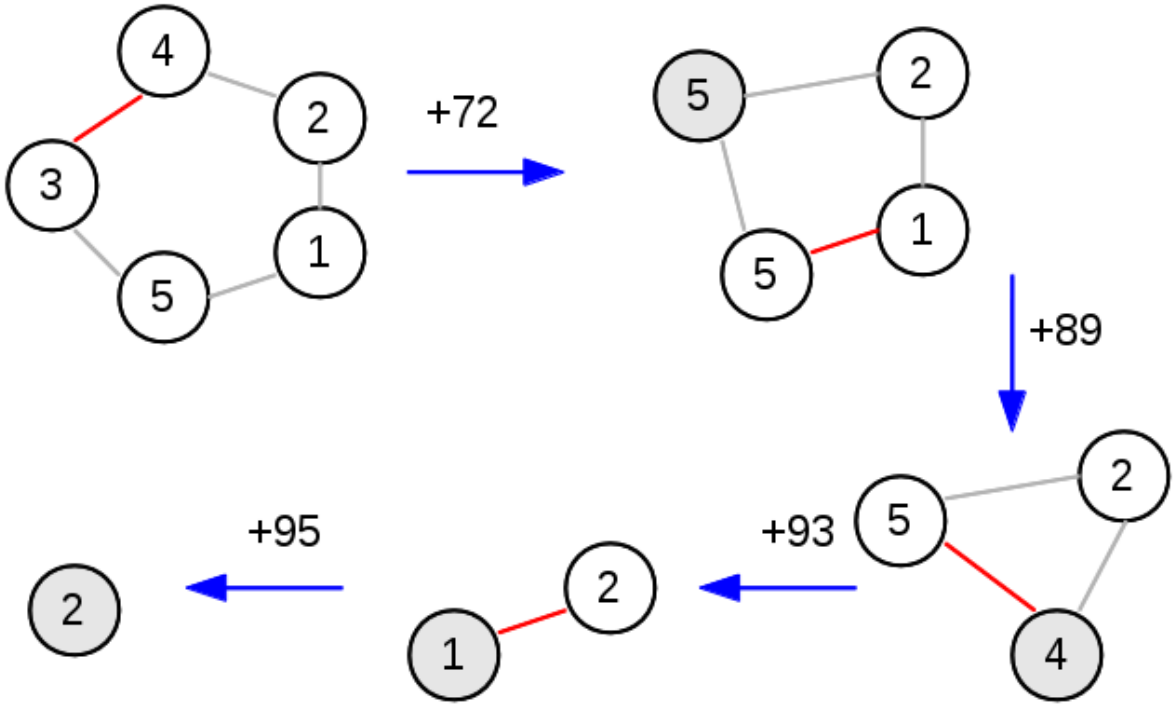
```
0 [ x  x  x  x  x 66][ x  x 93  x  x  x][ x  x  x  x  x  x][48  x  6 81 46  x][53  x 44 37  x 73][ x  x  x 99  x  x]
1 [ x  x 93  x  x  x][ x  x 98  x  x  x][ x  x 95  x  x  x][24  x  x 29 65  x][ x  x 22  7  x 22][80  x  x  x 89  x]
2 [ x  x  x  x  x  x][ x  x 95  x  x  x][ x  x  x  x  x 14][ x  x  x  x 95  x][ x  x  x 56  x  x][ x  x 77  x  x  x]
3 [48  x  6 81 46  x][24  x  x 29 65  x][ x  x  x  x 95  x][ x  x  x  x  x  x][ x  x  x  x  x 72][ x  x 50  x  x  x]
4 [53  x 44 37  x 73][ x  x 22  7  x 22][ x  x  x 56  x  x][ x  x  x  x  x 72][ x 94  x  x  x 76][63 93  x  x 57  x]
5 [ x  x  x 99  x  x][80  x  x  x 89  x][ x  x 77  x  x  x][ x  x 50  x  x  x][63 93  x  x 57  x][35 43  x  x  x  x]
```

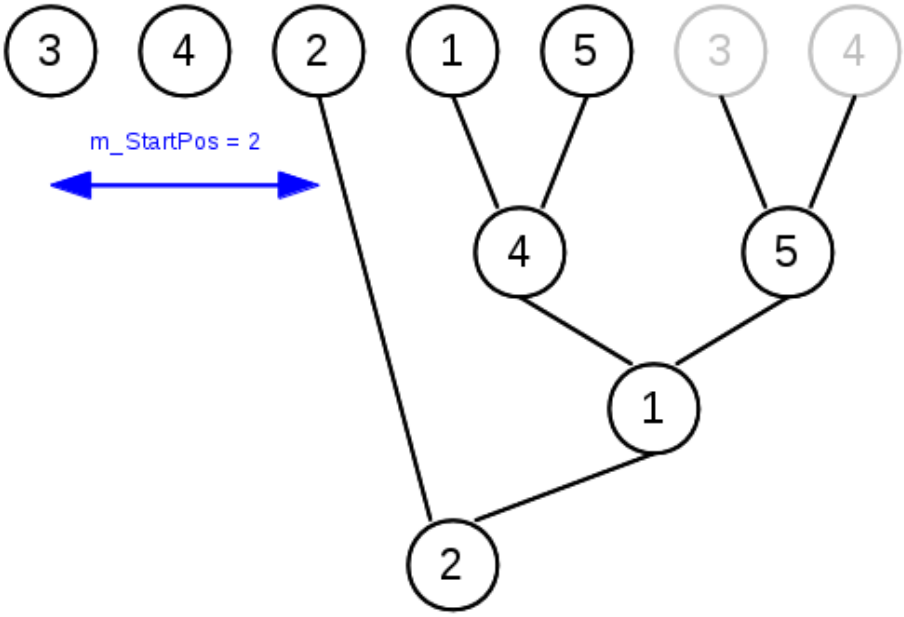
3D matice generátoru je zobrazená jako 2D, třetí index (uvolněná energie pro ten který výsledný kvark) se uplatní na hodnoty v hranaté závorce. Např. pro fúzi 5 + 4 se použije poslední řádek a předposlední sloupec, tedy možnosti jsou `[63 93 x x 57 x]`. Reakcí tedy lze získat kvarky 0, 1 nebo 4, uvolněná energie bude 63, 93, resp. 57. Všimněte si, že matice je symetrická podle diagonály, tedy fúze x + y a y + x dávají stejné možnosti.

```
Požadovaný finální produkt: 2
Palivový kruh: 3  4  2  1  5

Nalezený výsledek: energie 349, pozice 2, strom:
<2, 349>
 +-<2 @ 2>
 +-<1, 254>
    +-<4, 89>
    |  +-<1 @ 3>
    |  +-<5 @ 4>
    +-<5, 72>
       +-<3 @ 0>
       +-<4 @ 1>
```

Výpis stromu ukazuje listy v zápisu typu `<3 @ 0>`. Tento zápis říká. že v tomto místě je použit kvark 3 z palivového kruhu na původní pozici 0. Všimněte si, že pozice tvoří cyklickou sekvenci (čteme-li listy zleva doprava). Zápis vnitřních uzlů typu `< 1, 254 >` udává, že výsledným produktem fúze je kvark 1 a uvolněná energie je 254 (součet energie ze synů plus energie uvolněné poslední fúzí).





Odevzdávejte zdrojový kód s implementací požadovaných funkcí `optimizeEnergySeq` a `OptimizeEnergy`, s případnými dalšími podpůrnými funkcemi, které Vaše implementace potřebuje. Do Vaší implementace nevkládejte funkci `main` ani direktivy pro vkládání hlavičkových souborů. Funkci `main` a hlavičkové soubory lze ponechat pouze v případě, že jsou zabalené v bloku podmíněného překladu.

Využijte přiložené ukázkové soubory. Zdrojové soubory a Makefile lze použít pro lokální testování Vaší implementace. Celá implementace patří do souboru `solution.cpp`, dodaný soubor je pouze mustr. Pokud zachováte bloky podmíněného překladu, můžete soubor `solution.cpp` odevzdávat jako řešení úlohy. Ostatní zdrojové soubory umožňují načítat zadání generátorů a palivových kruhů a zobrazují Vaše výsledky. V adresáři data máte k dispozici sadu testovacích vstupů a jim odpovídajících výsledků.

Nápověda:

- Nejprve implementujte funkci `optimizeEnergySeq`, která analyzuje jeden palivový kruh. Tuto funkci budete následně potřebovat pro implementaci verze s více vlákny. Správnost implementace lze ověřit lokálně pomocí infrastruktury v přiloženém archivu. Až budete mít funkci lokálně otestovanou, můžete ji zkusit odevzdat na Progtest (pro tento pokus nechte `optimizeEnergy` s prázdným tělem). Takové řešení samozřejmě nedostane žádné body, ale uvidíte, zda správně projde prvním testem.
- Abyste zapojili co nejvíce jader, zpracovávejte několik kruhů najednou. Vyzvedněte je pomocí opakovaného volání `dispatcher` a okamžitě po analýze vracejte odpověď pomoci `engines`. Není potřeba dodržovat pořadí při odevzdávání. Pokud budete najednou zpracovávat pouze jeden kruh, nejspíše zaměstnáte pouze jedno vlákno a ostatní vlákna budou čekat bez užitku.
- Funkce `optimizeEnergy` je volaná opakovaně, pro různé vstupy. Nespoléhejte se na inicializaci globálních proměnných - při druhém a dalším zavolání budou mít globální proměnné hodnotu jinou. Je rozumné případné globální proměnné vždy inicializovat na začátku funkce `optimizeEnergy`. Ještě lepší je nepoužívat globální proměnné vůbec.
- Nepoužívejte mutexy a podmíněné proměnné inicializované pomocí `PTHREAD_MUTEX_INITIALIZER`, důvod je stejný jako v minulém odstavci. Použijte raději `pthread_mutex_init()`.
- Testovací prostředí samo o sobě nevytváří žádná vlákna, tedy funkce `optimizeEnergy` sama o sobě nemusí být reentrantní (může používat globální proměnné, s omezením výše).
- Popis generátorů a pole s palivovými kruhy alokovalo testovací prostředí. Testovací prostředí se také postará o jejich uvolnění (po převzetí funkcí `engines`). Jejich uvolnění tedy není Vaší starostí. Testovací prostředí dále maže `TSetup . m_Root` (zavolá na něj `delete` a následně nastaví `m_Root` na NULL). Váš program je ale zodpovědný za uvolnění všech ostatních prostředků, které si alokujete Vy sami.
- Palivové kruhy musíte načítat, zpracovávat a odevzdávat průběžně. Postup, kdy si všechny načtete do paměťových struktur a teprve pak je začnete zpracovávat, nebude fungovat.
- Volání `engines` je reentrantní, není potřeba je serializovat (obalovat mutexy). Každý kruh odevzdávejte právě 1x. Rozumné je volat funkci `engines` přímo z výpočetního vlákna, které pro daný kruh dokončilo analýzu a vyplnilo strukturu `TSetup`. Jak již bylo řečeno, `engines` je reentrantní, tedy není problém jej volat z více výpočetních vláken (pro různé palivové kruhy) najednou.
- Neukončujte funkci `optimizeEnergy` pomocí `exit`, `pthread_exit` a podobných funkcí. Pokud se funkce `optimizeEnergy` nevrátí do volajícího, bude Vaše implementace vyhodnocena jako nesprávná.
- Využijte přiložená vzorová data. V archivu jednak naleznete ukázku volání rozhraní (práce s ukazateli na funkce) a dále několik testovacích vstupů a odpovídajících výsledků.
- Nebojte se ukazatelů na funkce. Předaný parametr, např. `engines` použijte při volání jako každou jinou funkci:

```
void optimizeEnergy ( ..., void (*engines) ( const TRequest*, TSetup *), ... )
 {
   ...
   engines ( a, b );
   ...
```

```
 }
```

Pokud chcete volat funkci zprostředkovanou ukazatelem `engines` z jiné funkce než z `optimizeEnergy`, musíte si ukazatel předat:

```
void foo ( void )
 {
   // nefunguje, funkce s takovym jmenem (asi) v testovacim prostredi
   // neexistuje
   engines ( a, b );
 }
void bar ( void (* warpEngines) ( const TRequest*, TSetup *) )
 {
   warpEngines ( a, b ); // ok
 }
void optimizeEnergy ( ..., void (*engines) ( const TRequest*, TSetup *), ... )
 {
   ...
   foo ();
   bar ( engines );
   ...
 }
```

- V testovacím prostředí je k dispozici STL. Myslete ale na to, že ten samý STL kontejner nelze najednou zpřístupnit z více vláken. Více si o omezeních přečtěte např. na **C++ reference - thread safety.**
- Testovací prostředí je omezené velikostí paměti. Není uplatňován žádný explicitní limit, ale VM, ve které testy běží, je omezena 1 GiB celkové dostupné RAM. Úloha může být dost paměťově náročná, zejména pokud se rozhodnete pro jemné členění úlohy na jednotlivá vlákna. Pokud se rozhodnete pro takové jemné rozčlenění úlohy, možná budete muset přidat synchronizaci běhu vláken tak, aby celková potřebná paměť v žádný nepřesáhla nějaký rozumný limit. Pro běh máte garantováno, že Váš program má k dispozici nejméně 96 MiB pro Vaše data (data segment + stack + heap). Pro zvídavé - zbytek do 1GiB je zabraný běžícím OS, dalšími procesy, zásobníky Vašich vláken a nějakou rezervou.

Co znamenají jednotlivé testy:

Test algoritmu (sekvencni)
: Testovací prostředí opakovaně volá funkci `optimizeEnergySeq` pro různé palivové kruh (viz vzorová data `basic.txt`) a kontroluje vypočtené výsledky. Slouží pro otestování implementace Vašeho algoritmu. Funkce `optimizeEnergy` není v tomto testu vůbec volaná. Zároveň si na tomto testu můžete ověřit, zda Vaše implementace algoritmu je dostatečně rychlá.

Základní test/test několika/test mnoha thready,
: Testovací prostředí volá funkci `optimizeEnergy` pro různý počet vláken. Palivové kruhy jsou krátké a je jich málo.

Test zrychleni vypoctu
: Testovací prostředí spouští Vaši funkci pro ta samá vstupní data s různým počtem vláken. Měří se čas běhu (wall i CPU). S rostoucím počtem vláken by měl wall time klesat, CPU time mírně růst (vlákna mají možnost běžet na dalších CPU). Pokud wall time neklesne, nebo klesne málo (např. pro 2 vlákna by měl ideálně klesnout na 0.5, existuje určitá rezerva), test není splněn.

Busy waiting - pomaly dispatcher
: Do volání `dispatcehr` testovací prostředí vkládá uspávání vlákna (např. na 100 ms). Výpočetní vlákna tím nemají práci. Pokud výpočetní vlákna nejsou synchronizovaná blokujícím způsobem, výrazně vzroste CPU time a test selže.

Busy waiting - pomale engines
: Do volání `engines` je vložena pauza. Pokud je špatně blokováno vlákno dispatcher, výrazně vzroste CPU time. (Tento scénář je zde méně pravděpodobný.) Dále tímto testem neprojdete, pokud zbytečně serializujete volání `engines`.

Busy waiting - complex
: Je kombinací dvou posledně jmenovaných testů.

Test rozlozeni zateze
: Testovací zkouší výpočet pro jeden generátor a velké množství krátkých palivových kruhů. Výpočet by měl být rovnoměrně rozdělen na dostupná vlákna.

Test rozlozeni zateze (jemny)
: Testovací prostředí zkouší výpočet pro mnoho generátorů s jedním dlouhým palivovým kruhem. Pokud rozdělujete výpočetní zátěž podle schématu jeden kruh = jedno vlákno, bude v tomto testu zrychlení téměř nulové. Pokud chcete v tomto testu uspět, musíte Váš program navrhnout tak, aby bylo možné využít více vláken i při analýze jednoho kruhu. Test není povinný, ale jeho nezvládnutí znamená citelný bodový postih.

Test rozlozeni zateze (velmi jemny)
: Testovací prostředí zkouší výpočet pro jeden generátor a jeden dlouhý palivový kruh. Pokud rozdělujete výpočetní zátěž podle schématu jeden kruh x jeden generátor = jedno vlákno, bude v tomto testu zrychlení téměř nulové. Pokud chcete v tomto testu uspět, musíte Váš program navrhnout tak, aby bylo možné využít více vláken i při analýze jednoho kruhu. Takové rozdělení již není zcela jednoduché, proto je tento test bonusový.

Jak to vyřešit - pozor, SPOILER

Pokud se nechcete obrat o dobrý pocit, že jste úlohu vyřešili zcela sami, nečtěte dále.

- Základem řešení je oblíbený algoritmus CYK z ještě oblíbenějšího předmětu AAG.
- Kvarky představují neterminály, v gramatice tedy máme 6 neterminálů. Terminály zde nemáme, formálně si můžete představit kvarky v popisu palivového kruhu jako 6 terminálů. Jejich přepis na neterminály je (opět formálně) identita.

- Fúze typu A + B -> C je vlastně pravidlo gramatiky, navíc v Chomského normální formě.
- Popis generátoru je vlastně seznam pravidel gramatiky.

Další nápověda - SUPERSPOILER

- Výpočet energie získané při fúzi - to je vlastně syntetizovaný atribut. Takže při vyplňování tabulky CYKu si budeme pamatovat nejen neterminál, ale i hodnotu energie získané při fúzi podstromu. Pokud se k tomu samému neterminálu lze dostat více způsoby, vybereme ten způsob s vyšší uvolněnou energií.
- Algoritmus CYK pracuje pro řetězce, ale ne pro kruhy (kruhové řetězce). Lze jej ale snadno upravit pro kruh. Místo trojúhelníkové matice si pamatujte celý čtverec. CYK má v matici na pozici [i][j] seznam neterminálů, které jsou schopné generovat řetězec délky `i` počínaje pozicí `j`. Pro délku vstupního řetězce `l` jsou zakázané pozice, kde $i + j \geq l$ (indexujeme od 0), protože jsou "mimo" řetězec. Pokud je povolíme a index `j` budeme počítat modulo `l`, získáme přesně kruh. Volba kruhu není samoúčelné ztížení úlohy. Ve variantě s kruhem se úloha snáze paralelizuje do více vláken.
- Posunutí v kruhu (`m_StartPos`) je pro klasický CYK vždy 0. Pro náš upravený kruhový CYK je to pozice, kde v poslední řádce nalezneme požadovaný finální kvark s nejvyšší uvolněnou energií.

Vzorová data: **Download**

Referenční řešení

3	**15.03.2014 13:20:15**	**Download**
Stav odevzdání:	Ohodnoceno	
Hodnocení:	33.0000	

- **Hodnotitel: automat**
  - Program zkompilován
  - Test 'Test algoritmu (sekvencni)': Úspěch
    - Dosaženo: 100.00 %, požadováno: 100.00 %
    - Celková doba běhu: 0.247 s (limit: 10.000 s)
    - Úspěch v závazném testu, hodnocení: 100.00 %
  - Test 'Zakladni test (1 thread)': Úspěch
    - Dosaženo: 100.00 %, požadováno: 80.00 %
    - Celková doba běhu: 0.255 s (limit: 9.753 s)
    - Úspěch v závazném testu, hodnocení: 100.00 %
  - Test 'Test nekolika thready': Úspěch
    - Dosaženo: 100.00 %, požadováno: 80.00 %
    - Celková doba běhu: 0.072 s (limit: 9.498 s)
    - CPU time: 0.335 s (limit: 9.497 s)
    - Úspěch v závazném testu, hodnocení: 100.00 %
  - Test 'Test mnoha thready': Úspěch
    - Dosaženo: 100.00 %, požadováno: 80.00 %
    - Celková doba běhu: 0.061 s (limit: 9.426 s)
    - CPU time: 0.392 s (limit: 9.162 s)
    - Úspěch v závazném testu, hodnocení: 100.00 %
  - Test 'Test zrychleni vypoctu': Úspěch
    - Dosaženo: 100.00 %, požadováno: 50.00 %
    - Celková doba běhu: 1.390 s (limit: 9.365 s)
    - CPU time: 3.036 s (limit: 8.770 s)
    - Úspěch v závazném testu, hodnocení: 100.00 %
  - Test 'Busy waiting test (pomaly dispatcher)': Úspěch
    - Dosaženo: 100.00 %, požadováno: 50.00 %
    - Celková doba běhu: 1.265 s (limit: 10.000 s)
    - Úspěch v nepovinném testu, hodnocení: 100.00 %
  - Test 'Busy waiting test (pomale engines)': Úspěch
    - Dosaženo: 100.00 %, požadováno: 50.00 %
    - Celková doba běhu: 0.737 s (limit: 8.735 s)
    - Úspěch v nepovinném testu, hodnocení: 100.00 %
  - Test 'Busy waiting test (complex)': Úspěch
    - Dosaženo: 100.00 %, požadováno: 50.00 %
    - Celková doba běhu: 1.964 s (limit: 7.998 s)
    - Úspěch v nepovinném testu, hodnocení: 100.00 %
  - Test 'Test rozlozeni zateze (jemny)': Úspěch
    - Dosaženo: 100.00 %, požadováno: 75.00 %
    - Celková doba běhu: 0.521 s (limit: 6.034 s)
    - CPU time: 1.861 s (limit: 9.542 s)
    - Úspěch v nepovinném testu, hodnocení: 100.00 %
  - Test 'Test rozlozeni zateze (jeste jemnejsi)': Neúspěch
    - Dosaženo: 50.00 %, požadováno: 100.00 %
    - Celková doba běhu: 1.526 s (limit: 10.000 s)
    - Neúspěch v bonusovém testu, hodnocení: Bonus nebude udělen
    - Nesprávný výstup

- Celkové hodnocení: 100.00 % (= 1.00 * 1.00 * 1.00 * 1.00 * 1.00 * 1.00 * 1.00 * 1.00 * 1.00)
- Celkové procentní hodnocení: 100.00 %
- Bonus za včasné odevzdání: 3.00
- Celkem bodů: 1.00 * (30.00 + 3.00) = 33.00

		Celkem	Průměr	Maximum	Jméno funkce
SW metriky:	Funkce:	**11**	--	--	--
	Řádek kódu:	**266**	**24.18 ± 17.25**	**52**	`workerRoutine(void *)`
	Cyklomatická složitost:	**61**	**5.55 ± 3.85**	**12**	`fillCYKField(const int,const int)`

2	**13.03.2014 00:27:53**	**Download**
Stav odevzdání:	Ohodnoceno	
Hodnocení:	0.0000	

- **Hodnotitel: automat**
  - Chyba při kompilaci v režimu 'pedantic' - 10% penalizace
  - Test 'Test algoritmu (sekvencni)': Úspěch
    - Dosaženo: 100.00 %, požadováno: 100.00 %
    - Celková doba běhu: 0.243 s (limit: 10.000 s)
    - Úspěch v závazném testu, hodnocení: 100.00 %
  - Test 'Zakladni test (1 thread)': Úspěch
    - Dosaženo: 92.31 %, požadováno: 80.00 %
    - Celková doba běhu: 0.236 s (limit: 9.757 s)
    - Úspěch v závazném testu, hodnocení: 92.31 %
    - Nesprávný výstup
  - Test 'Test nekolika thready': Program provedl neplatnou operaci a byl ukončen (Segmentation fault/Bus error/Memory limit exceeded/Stack limit exceeded)
    - Celková doba běhu: 0.121 s (limit: 9.521 s)
    - CPU time: 0.351 s (limit: 9.521 s)
    - Neúspěch v závazném testu, hodnocení: 0.00 %
  - Celkové hodnocení: 0.00 % (= (1.00 * 0.92 * 0.00) * 0.9)
- Celkové procentní hodnocení: 0.00 %
- Bonus za včasné odevzdání: 3.00
- Celkem bodů: 0.00 * (30.00 + 3.00) = 0.00

		Celkem	Průměr	Maximum	Jméno funkce
SW metriky:	Funkce:	**9**	--	--	--
	Řádek kódu:	**214**	**23.78 ± 13.87**	**44**	`fillCYKField(const int,const int)`
	Cyklomatická složitost:	**51**	**5.67 ± 3.92**	**12**	`fillCYKField(const int,const int)`

1	**12.03.2014 14:56:23**	**Download**
Stav odevzdání:	Ohodnoceno	
Hodnocení:	0.0000	

- **Hodnotitel: automat**
  - Chyba při kompilaci v režimu 'pedantic' - 10% penalizace
  - Test 'Test algoritmu (sekvencni)': Úspěch
    - Dosaženo: 100.00 %, požadováno: 100.00 %
    - Celková doba běhu: 0.246 s (limit: 10.000 s)
    - Úspěch v závazném testu, hodnocení: 100.00 %
  - Test 'Zakladni test (1 thread)': Neúspěch
    - Dosaženo: 50.00 %, požadováno: 80.00 %
    - Celková doba běhu: 0.000 s (limit: 9.754 s)
    - Neúspěch v závazném testu, hodnocení: 0.00 %
    - Nesprávný výstup
  - Celkové hodnocení: 0.00 % (= (1.00 * 0.00) * 0.9)
- Celkové procentní hodnocení: 0.00 %
- Bonus za včasné odevzdání: 3.00
- Celkem bodů: 0.00 * (30.00 + 3.00) = 0.00

		Celkem	Průměr	Maximum	Jméno funkce
SW metriky:	Funkce:	**7**	--	--	--

Řádek kódu:	**150**	**21.43 ± 13.77**	**44**	`fillCYKField(const int,const int)`
Cyklomatická složitost:	**39**	**5.57 ± 4.30**	**12**	`fillCYKField(const int,const int)`