Využití MPI při distribuovaných výpočtech

Vilém Vychodil

21. července 2001

Abstrakt

Tento referát se zabývá využitím standardisovaného rozhraní MPI — Message Passing Interface při distribuovaných výpočtech. Pojmem distribuovaný výpočet je obvykle myšlen mechanismus, umožňující řešit jednu úlohu na několika uzlech bez možnosti přímo sdílet paměť. V následujícím textu, který by měl sloužit jako vysvětlující doprovod k předloženým ukázkovým programům, bych chtěl zdůraznit některé aspekty programování distribuovaných numerických úloh.

Message Passing Interface

Standard MPI popisuje knihovní rozhraní pro zasílání zpráv. Přitom plně respektuje heterogenní prostředí, ve kterém může být celý distribuovaný systém nasazen. Zakrytí hardwaru je téměř úplné, programátor se rozhodně nemusí zabývat zbytečnostmi, jako je *little a big endian* a podobně. Standard definuje základní datové typy, mezi které patří `integer`, `double`, `long double` a jejich komplexní ekvivalenty.

Pro implementaci demonstračních úloh jsem si zvolil operační systém Linux, protože vyniká rychlostí přepínání úloh a je díky své „štíhlosti“ vhodný pro nasazení v distribuovaném prostředí. Volně šiřitelná implementace standardu MPI, je součástí všech významných distribucí Linuxu. Distribuce Debian/GNU Linux obsahuje navíc několik známých knihoven funkcí, které byly portovány pro MPI. Například `scalapack` je `linpack`, umožňující snadné programování distribuovaných numerických úloh. Pro detaily viz `http://www.debian.org`.

Aproximace čísla π

Neformálně řečeno, snažíme se aproximovat π numerickou kvadraturou

$$\pi = \int_0^1 \frac{4}{1+x^2}\,\mathrm{d}x.$$

Při výpočtu můžeme použít například obdélníkovou metodu. Předpokládejme, že interval $\langle 0,1\rangle$ rozdělíme na n podintervalů. Výslednou hodnota je aproximována součtem obsahů obdélníků, jejichž rozměry jsou $\frac{1}{n}$ a $\frac{4}{1+t^2}$, kde t se středem obdélníka. V tomto případě evidentně nic nebrání tomu, abychom na každém výpočetním uzlu počítali obsah jen jistého množství obdélníků a výsledky potom sečetli. Máme-li k uzlů, může probíhat současně k součtů n/k obsahů. V tomto případě je v zásadě nutné zajistit jen celkový součet.

Viz zdroják `estimate-pi.c`. Za pozornost v něm stojí několik věcí. Voláním `MPI_Init (&argc, &argv)` se inicialisuje MPI, stejně tak na konci se používá volání `MPI_Finalize ()`, které ukončí jednotlivé procesy. Jednotlivé instance jsou na vzdálených uzlech spouštěny pomocí služeb `rsh` nebo `ssh`. Při spuštění programy je dobré předat soubor, v němž jsou uvedeny stroje, na kterých bude program spuštěn. Soubor obsahuje řádky v následujícím tvaru.

```
stroj číslo binárka uživatel
```

Kde *číslo* je buďto nula, to jest na mašině se kromě lokální instance již nic nespouští nebo 1.

K samotnému předávání zpráv existuje řada funkcí, nejčastěji používané jsou `MPI_Bcast` a `MPI_Reduce`. Funkce `MPI_Bcast` slouží ke všesměrovému zaslání signálu od jednoho procesu — koordinátora, k ostatním procesům. Funkce `MPI_Reduce` slouží k redukci více hodnot z různých uzlů na jednu hodnotu pro koordinátora, typicky součet, součin atd. Viz zdrojový kód.

Jacobiho iterační metoda

Idea je analogická. Máme danou soustavu n lineárních rovnic, z i-té rovnice vyjádříme neznámou x_i, jinak řečeno, iterativní formule je tvaru

$$\vec{y} = -\mathbf{D}^{-1}(\mathbf{L}+\mathbf{U})\vec{x} + \mathbf{D}^{-1}\vec{b},$$

to všichni známe ze základního kursu numeriky. Tím získáme (možná lepší) aproximaci řešení. Jak je na první pohled vidět, násobení matice vektorem, které je tu stěžejní, lze provádět opět nezávisle. To jest, každý uzel může násobit jen určité sloupce a sečtením částečných výsledků obdržíme hledaný vektor. Pro detaily viz `linsolve.c`.