PGI CUDA Fortran

rozšíření PGI Fortranu pro GPU Computing zahrnující
– zápis procedur určených k běhu na GPU (kernel subroutines: atribut global, device procedures: atribut device)
– konfigurovatelné volání kernelu (call MyKernel <<<grid,block>>> (parameters))
– rozvrhování proměnných v paměti GPU (atributy device, constant, shared; atribut pinned)
– alokace paměti a přesuny dat mezi hostitelem a GPU (allocate a přiřazovací příkazy)
– volání funkcí CUDA Runtime API
– modul cudafor s popisem typů (dim3), proměnných (threadidx, blockidx, blockdim,griddim) a rozhraní API funkcí

Příprava, konfigurace a překlad zdrojového kódu
instalace driveru s CUDA (www.nvidia.com)
testovací utilita pgaccelinfo
překlad: pgfortran -Mcuda file.f90 nebo pgfortran file.cuf
-Mcuda=11 pro nižší Compute Capability (REAL(8) budou „demoting to float")
-Mcuda=emu pro emulační mód (pomocí OpenMP tasks)
kombinace s OpenMP a MPI možná při více GPU

Kernel
– podprogram (subroutine) s atributem global volaný z hostitele a prováděný paralelními vlákny na GPU
– př. ATTRIBUTES(GLOBAL) SUBROUTINE MyKernel(a,n)
– jiné atributy: host (default) pro proceduru určenou k běhu na hostiteli
device pro proceduru určenou k běhu na GPU, volanou z kernelu nebo jiné device procedury

– kernel musí být SUBROUTINE, nelze RECURSIVE, PURE ani ELEMENTAL, nesmí obsahovat CONTAINS
a být vnořen jinde než v modulu, žádné volitelné argumenty
– nesmí obsahovat pole předpokládaného tvaru (a(:)) a data s atributy POINTER a ALLOCATABLE,
může obsahovat pole předpokládané velikosti (a(*))
– nesmí obsahovat příkazy vstupu a výstupu, STOP, PAUSE
– může obsahovat volání procedur s atributem device, překladač však může odmítnout (podmínka: inlining)
– explicitní rozhraní je obvykle nezbytné, tj. vnoření do modulu vhodné

– možnost zjištění pořadového čísla vlákna v „3D" bloku pomocí proměnných threadidx a blockdim typu dim3
TYPE dim3 ; INTEGER(4) x,y,z ; END TYPE ; TYPE(dim3) threadidx, blockdim
– převod do 1D (první položka se mění nejrychleji, položky začínají od 1):
1D … threadidx%x
2D … threadidx%x+blockdim%x*(threadidx%y-1)
3D … threadidx%x+blockdim%x*(threadidx%y-1+blockdim%y*(threadidx%z-1))
– podobně možnost zjištění pořadového čísla bloku v „3D" mřížce pomocí proměnných blockidx a griddim typu dim3

– parametry kernelu mají
atribut device: skutečný argument je umístěn v device global memory (atribut je patrně default)
atribut value: skutečný argument je v host memory a musí být přenesen hodnotou (lze jen pro skaláry)
př. REAL(DP),DEVICE :: a(*)
INTEGER,VALUE :: n

– povolené datové typy:
INTEGER(1,2,4,8), LOGICAL(1,2,4,8), REAL(4,8), COMPLEX(4,8), CHARACTER(1) a typy z nich odvozené
– povolené fortranské funkce a procedury:
abs, aimag, aint, …, min, max, …, acos, asin, atan, …, all, any, count, maxloc, maxval, sum, …,
dot_product, matmul, random_seed, random_number
– synchronizace vláken v bloku:
CALL syncthreads( )

Volání kernelu

– kernel provádějí vlákna uspořádaná do bloků (prováděných na jednom SM, synchronizovatelných, s přístupem do sdílené paměti SM) a bloky uspořádané do mřížky (bloky běží nezávisle, nejsou synchronizovatelné a komunikovat mohou jen přes sdílenou device memory), celková velikost bloku je omezena na 512 vláken (CC 1.1 i 1.3)
– konfigurace bloku i mřížky může být 1D (pomocí integer skaláru) až 3D (pomocí proměnné typu dim3), velikost třetí dimenze mřížky je povinně 1
– př. INTEGER :: block=512,grid=2048 ; call MyKernel <<<grid,block>>> (parameters)
TYPE(dim3) :: block=dim3(512,1,1),grid=dim3(2048,1,1) ; call MyKernel <<<grid,block>>> (parameters)
– další parametry: <<grid,block,bytes,streamid>>
pro dynamickou alokaci sdílené paměti SM pro každý blok (integer bytes, default 0 B) a pro využití proudových front (stream queues), běžících asynchronně (integer streamid, default 0)
– volání kernelu je asynchronní, tj. host pokračuje bez čekání na dokončení kernelu, vynutit synchronizaci lze voláním CUDA (integer) funkce cudaThreadSynchronize(), v praxi stačí (implicitně synchronizující) přenos dat z device do host memory

Rozvrhování proměnných v paměti GPU

– proměnná určená k umístění v paměti GPU je popsána pomocí atributů device, constant nebo shared
– př. REAL,DEVICE :: a(1000) ; ATTRIBUTES(CONSTANT) :: pi
– proměnné s atributy device a constant jsou v hostitelské proceduře volitelně inicializovány přiřazovacím příkazem, odeslány kernelu jako skutečný argument a po ukončení kernelu se jejich obsah přiřazovacím příkazem přenese zpět do paměti hostitele
– proměnné shared jsou lokální v kernelu a s ukončením kernelu zanikají, mohou být sdíleny vlákny v bloku
– proměnné bez atributů jsou vytvořeny v paměti hostitele
– atribut device: pro umístění proměnné v device global memory
lze pro modulové proměnné, nelze pro položky odvozených typů, nelze s atributy POINTER, TARGET a SAVE,
proměnná v hostitelské proceduře může být použita jen v popisech, při alokaci a dealokaci, jako skutečný nebo formální argument a v přiřazovacích příkazech pro přenos mezi hostitelem a GPU
– atribut constant: pro umístění proměnné v device constant memory space (lokální cache SM pro konstanty 8 KB)
může být modifikována hostitelem, nemůže být modifikována kernelem
lze pro modulové proměnné, nelze s atributem ALLOCATABLE
– atribut shared: pro umístění proměnné v share memory space (lokální paměť SM 16 KB)
lokální proměnná kernelu (nebo device procedury), nelze s atributem ALLOCATABLE
dynamická alokace sdílené proměnné: při deklaraci př. REAL,SHARED :: x(*) je poli x přidělena sdílená paměť o velikosti určené argumentem bytes z volání kernelu
– atribut value: skaláry v paměti hostitele nemohou být do kernelu (který potřebuje data umístěna v device memory) předávány (implicitně) odkazem, ale hodnotou, tj. explicitním uvedením atributu value
– skaláry a proměnné CUDA-předdefinovaných typů ukládány v registrech, pole v device global memory

Alokace paměti a přesuny dat mezi hostitelem a GPU

– pole umístěná v device memory (atribut device) mohou být alokovatelná a alokují se v hostitelské proceduře běžným ALLOCATE, device atribut mohou mít i automatická pole
– alokovatelná pole v paměti hostitele mohou mít atribut pinned pro alokování v zamčené paměti, je-li k dispozici, přenosy takových polí na GPU mohou být rychlejší
– životnost alokovatelných polí s atributem device je řízena pravidly Fortranu, lze však používat i alokační funkce CUDA API s životností polí podle pravidel C
– přesuny dat (s atributy device nebo constant) mezi hostitelem a GPU se dějí v hostitelské proceduře běžnými přiřazovacími příkazy (co možná nejjednoduššími)
– výrazy v hostitelské proceduře zahrnující data uložená na hostiteli i GPU jsou s omezeními možné, ale riskantně neefektivní
– pro přesuny lze používat i funkce CUDA API