CAP-387(2016) – Tópicos Especiais em Computação Aplicada: Construção de Aplicações Massivamente Paralelas

Aula 32: Interface para Ferramentas em MPI

Celso L. Mendes, Stephan Stephany

LAC / INPE

Emails: celso.mendes@inpe.br, stephan.stephany@inpe.br

Interfaces Gerais em MPI

- MPI-1,2:
  - Interface para *profiling*: para toda função *MPI_Xyz()*, existe uma função similar *PMPI_Xyz()*, que permite ao programador "capturar" e tratar as chamadas MPI
- MPI-3:
  - Nova interface (adicional), para ferramentas
  - Permite obter informações de desempenho e de depuração
    - Exemplos:
      a) Qual foi o tempo de espera num *MPI_Recv*?
      b) Quantas mensagens chegaram e estão em espera na fila?

Interface para Ferramentas em MPI

- MPI-3:
  - Implementações de MPI-3 podem manter tal informação
  - Tipicamente, tais informações são úteis tanto para o usuário de MPI como para os desenvolvedores da biblioteca
  - Informações estão em dois tipos principais de variáveis internas mantidas pela biblioteca MPI:
    a) Variáveis de controle (cvar): definem o comportamento da biblioteca MPI
    b) Variáveis de desempenho (pvar): quantificam aspectos da biblioteca
  - MPI-3 padroniza o acesso a tais variáveis, quando existirem
    - OBS: Interface definida apenas para programas em C

Interface para Ferramentas em MPI

- Características comuns aos dois tipos de variáveis:
  - Padrão MPI-3 não obriga a existência de variáveis pré-definidas
  - Variáveis são associadas a um número inteiro ≥ 0
  - O número de variáveis disponíveis pode aumentar (mas nunca diminuir) durante a execução de um programa MPI
    - Ex: Ao se carregar dinamicamente uma função, novas variáveis podem passar a existir no programa
  - Variáveis são acessadas através de um *handle* opaco
    - Permite à biblioteca ocultar informação não relevante
    - Exemplos de *handle*:
      - Ponteiro para a variável
      - Ponteiro para uma estrutura contendo a variável

Inicialização e Término da Interface

- Inicialização da interface:

  *int MPI_T_init(int required, int *provided)*

  - Argumentos:
    - *required*: nível de suporte a *thread* desejado
    - *provided*: nível de suporte a *thread* realmente oferecido
  - Pode ser chamada antes ou depois de *MPI_Init*

  Níveis de suporte a thread possíveis:

  - MPI_THREAD_SINGLE: apenas um thread vai ser executado
  - MPI_THREAD_FUNNELED: todas as chamadas a MPI_T serão feitas a partir do thread principal
  - MPI_THREAD_SERIALIZED: vários threads podem chamar MPI_T, mas apenas um de cada vez
  - MPI_THREAD_MULTIPLE: múltiplos threads podem chamar MPI_T, sem restrições

Inicialização e Término da Interface

- Término da interface:

  int MPI_T_finalize(void)
  - Pode ser chamada antes ou depois de *MPI_Finalize()*
- Valores de Retorno: sucesso/erro
  - Sucesso: MPI_SUCCESS
  - Erro: MPI_T_ERR_NOT_INITIALIZED
    - Programa não precisa ser abortado em caso de erro
- Combinação *T_init/T_finalize*
  - Pode ser usada várias vezes num mesmo programa

Variáveis de Controle

- Objetivo:
  - Permitir ao usuário influenciar a forma de operação da biblioteca de MPI

  Exemplo: definição do limiar dos protocolos *eager/rendezvous*
- Introspecção:
  - Ao invés de se basear em documentação, a própria biblioteca pode informar sobre a funcionalidade disponível:
    - Função para informar o número de variáveis disponíveis
    - Função para reportar detalhes de cada variável de controle

Interface para Variáveis de Controle

- Obter o número de variáveis disponíveis:

*int MPI_T_cvar_get_num(int *num)*

- Obter informações sobre uma variável com certo índice:

*int MPI_T_cvar_get_info(int cvar_index, char *name, int *name_len, int *verbosity, MPI_Datatype *datatype, MPI_T_enum *enumtype, char *desc, int *desc_len, int *binding, int *scope)*

- Obter o índice de uma certa variável:

*int MPI_T_cvar_get_index(const char *name, int *cvar_index)*

Detalhes de Variáveis de Controle

verbosity:

- Define a "audiência" da variável; valores são do tipo MPI_T_VERBOSITY_<user>_<detail>, onde <user> pode ser {USER,TUNER, MPIDEV} e <detail> é {BASIC,DETAIL,ALL}

binding:

- Especifica se a variável se refere a um objeto MPI específico ou não: MPI_T_BIND_T_COMM × MPI_T_BIND_NO_OBJECT

scope:

- Indica se a variável pode ser modificada e, se puder, se é por um único processo ou por um grupo de processos

Acesso a Variáveis de Controle

- Alocar um *handle* para acesso à variável:

*int MPI_T_cvar_handle_alloc(int cvar_index, void *obj_handle, MPI_T_cvar_handle *handle, int *count)*

- Liberar um *handle*:

*int MPI_T_cvar_handle_free(MPI_T_cvar_handle *handle)*

- Ler uma variável de controle:

*int MPI_T_cvar_read(MPI_T_cvar_handle handle, void *buf)*

- Escrever uma variável de controle:

*int MPI_T_cvar_write(MPI_T_cvar_handle handle, const void *buf)*

Variáveis de Controle - Exemplo

- Exemplo de uso:

```
MPI_T_init_thread(required, &provided);
MPI_T_cvar_get_num(*num_cvar);
printf("%d MPI Control Variables\n", num_cvar);
for (i=0; i<num_cvar; i++) {
        err = MPI_T_cvar_get_info(I, name, sizeof(name),
                &verbosity, &datatype, &enumtype, desc,
                sizeof(desc), &binding, &scope);
        printf("\t%-32s\t%s\n", name, desc);
}
MPI_T_finalize()
```

Variáveis de Controle - Saída

- Saída da execução do exemplo no Santos Dumont:

60 MPI Control Variables

MPIR_CVAR_DEBUG_HOLD If true, causes processes to wait in MPI_Init and MPI_Initthread for a debugger to be attached. Once the debugger has attached, the variable 'hold' should be set to 0 in order to allow the process to continue (e.g., in gdb, "set hold=0").

MPIR_CVAR_ERROR_CHECKING If true, perform checks for errors, typically to verify valid inputs to MPI routines. Only effective when MPICH is configured with --enable-error-checking=runtime .

. . .

MPIR_CVAR_CH3_EAGER_MAX_MSG_SIZE This cvar controls the message size at which CH3 switches from eager to rendezvous mode.

. . .

Variáveis de Desempenho

- Objetivo:
  - Permitir ao usuário acesso uniforme às variáveis de desempenho mantidas pela biblioteca MPI
  - Variáveis de desempenho: *performance variables* (pvar)
- Interface:
  - Similar à utilizada com as variáveis de controle:
    - *MPI_T_pvar_get_num:* obtém número de variáveis
    - *MPI_T_pvar_get_info:* obtém informações de uma variável
    - *MPI_T_pvar_get_index:* obtém índice de uma variável

Variáveis de Desempenho - Interface

- Obter o número de variáveis disponíveis:

*int MPI_T_pvar_get_num(int *num)*

- Obter informações sobre uma variável com certo índice:

*int MPI_T_pvar_get_info(int pvar_index, char *name, int *name_len, int *verbosity, int *var_class, MPI_Datatype *datatype, MPI_T_enum *enumtype, char *desc, int *desc_len, int *binding, int *isreadonly, int *iscontinuous, int *isatomic)*

- Obter o índice de uma certa variável:

*int MPI_T_pvar_get_index(const char *name, int var_class, int *pvar_index)*

Detalhes de Variáveis de Desempenho

Diferenças em relação às variáveis de controle:

var_class:

Classe da variável (nível, contador, relógio, marca d'água, etc)

readonly:

Se *false*, indica que a variável pode ser escrita pelo usuário

continuous:

Se *true*, indica que a variável é modificada continuamente pela bib. MPI
Caso contrário, ela pode ser iniciada (com *MPI_T_pvar_start*) ou parada (com *MPI_T_pvar_stop*) livremente pelo usuário

atomic:

Indica se a variável pode receber operações atômicas

Uso das Variáveis de Desempenho

- Para se usar estas variáveis, é preciso definir *sessões* (partes do programa) onde as variáveis são acessadas
  - Criação de sessões: *MPI_T_pvar_session_create()*
- Uma vez criada a sessão, as variáveis de desempenho podem ser acessadas através de um *handle*, tal qual as variáveis de controle
  - Porém, agora o tipo do *handle* é *MPI_T_pvar_handle*

Acesso a Variáveis de Desempenho

- Criar uma sessão para acesso à variável:

*int MPI_T_pvar_session_create(MPI_T_pvar_session *session)*

- Liberar uma sessão:

*int MPI_T_pvar_session_free(MPI_T_pvar_session *session)*

- Iniciar uma certa variável não-continua:

int MPI_T_pvar_start(MPI_T_pvar_session session, MPI_T_pvar_handle handle)

- Parar uma certa variável não-continua (para ler a variável):

int MPI_T_pvar_stop(MPI_T_pvar_session session, MPI_T_pvar_handle handle)

Acesso a Variáveis de Desempenho

- Alocar um *handle* para acesso à variável:

*int MPI_T_pvar_handle_alloc(MPI_T_pvar_session session, int pvar_index, void *obj_handle, MPI_T_pvar_handle *handle, int *flag)*

- Liberar um *handle*:

*int MPI_T_pvar_handle_free(MPI_T_pvar_session session, MPI_T_pvar_handle *handle)*

- Ler uma variável de desempenho:

*int MPI_T_pvar_read(MPI_T_pvar_session session, MPI_T_pvar_handle handle, void *buf)*

- Escrever uma variável de desempenho:

*int MPI_T_pvar_write(MPI_T_pvar_session session, MPI_T_pvar_handle handle, const void *buf)*

Variáveis de Desempenho - Exemplo

```
int provided, lcount, err, pidx, isContinuous; double qtimestart, qtime;
MPI_T_enum enumtype;  MPI_Datatype datatype;
MPI_T_pvar_session session;  MPI_T_pvar_handle handle;
MPI_Init_thread(0, 0, MPI_THREAD_SINGLE, &provided);
err = MPI_T_init_thread(MPI_THREAD_SINGLE, &provided);
if (err) MPI_Abort(MPI_COMM_WORLD, 0);
err = MPI_T_pvar_get_index("time_matching_unexpectedq", MPI_T_PVAR_CLASS_TIMER, &pidx);
if (err != MPI_SUCCESS) MPI_Abort(MPI_COMM_WORLD, 0);
err = MPI_T_pvar_get_info(pidx,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,&isContinuous,NULL);
err = MPI_T_pvar_session_create(&session);
err = MPI_T_pvar_handle_alloc(session, pidx, NULL, &handle, &lcount);
err = MPI_T_pvar_read(session, handle, &qtimestart);
if (!isContinuous) err = MPI_T_pvar_start(session, handle);
TestProgram();
if (!isContinuous) err = MPI_T_pvar_stop(session, handle);
err = MPI_T_pvar_read(session, handle, &qtime);
printf("Time searching unexpected queue = %e\n", qtime-qtimestart);
MPI_T_pvar_handle_free(session, &handle);
MPI_T_pvar_session_free(&session);
MPI_T_finalize(); MPI_Finalize();
```

Função com muita troca de mensagens

Fonte: Gropp et al *Using Advanced MPI*